算法艺术与信息学竞赛

- 《算经竞赛入门经典》的练习伴侣
- 实用编程技巧与C++11应用介绍
- 简洁、清晰、高效的C++示例代码
- 近期高质量竞赛真题分类选解
- 近期高质量竞赛真题选译



陈 锋◎编著

清华大学出版社

算法艺术与信息学竞赛

算法竞赛入门经典——习题与解答

陈　锋　编著

清华大学出版社

北　京

内 容 简 介

本书是在《算法竞赛入门经典（第 2 版）》的基础上，延伸出来的一本习题与解答图书，它把 C++语言、算法和解题有机地结合在一起，淡化理论，注重学习方法和实践技巧，是一本算法竞赛的入门和提高教材。

本书分为 5 章。第 1 章是各种编程训练技巧以及 C++11 语法特性的简单介绍。第 2 章精选了一部分《算法竞赛入门经典（第 2 版）》的习题进行分析、解答。第 3 章是 ACM/ICPC 比赛真题分类选解，挑选了近些年 ACM/ICPC 比赛中较有价值的题目进行分析并解答。第 4～5 章是比赛真题选译，整理并翻译了近几年来各大区域比赛中笔者认为值得学习训练的比赛真题。

如果你对算法感兴趣，如果你是一名程序员或即将成为一名程序员，如果你想大幅提升自己的算法思维能力，如果你有志于参加 ACM/ICPC、NOIP、NOI 等竞赛，那就来吧！本书将为你推开一扇算法世界的大门！

图书在版编目（CIP）数据

算法竞赛入门经典——习题与解答/陈锋编著．—北京：清华大学出版社，2018
（算法艺术与信息学竞赛）
ISBN 978-7-302-47658-0

Ⅰ．①算…　Ⅱ．①陈…　Ⅲ．①计算机算法-题解　Ⅳ．①TP301.6-44

中国版本图书馆 CIP 数据核字（2017）第 154508 号

责任编辑：贾小红
封面设计：刘　超
版式设计：刘艳庆
责任校对：马子杰
责任印制：沈　露

出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
　　社 总 机：010-62770175　　**邮　　购**：010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：28　**字　　数**：690 千字
版　　次：2018 年 1 月第 1 版　**印　　次**：2018 年 1 月第 1 次印刷
印　　数：1～3500
定　　价：69.80 元

产品编号：066737-01

推 荐 序

《算法竞赛入门经典（第 2 版）》问世时，我的心里终于放下了一块大石头。两年多来，因为工作繁忙，我几乎没有再碰及算法竞赛，各种国内外赛事的命题、培训邀请，我都一一回绝。然而，初心未变，我很希望这套丛书能够继续下去，能帮助到更多的人。所以，当这本《算法竞赛入门经典——习题与解答》终于完稿时，我的心情比许多新老读者还要激动。

这无疑是一本很特别的书！陈锋之前虽然参与了《算法竞赛入门经典——训练指南》的编写工作，但大家肯定不知道，他竟然是一名从没有参加过 NOIP 或者 ACM/ICPC 的“非职业选手”，甚至连计算机编程，他都是在大学毕业之后凭着一腔热情和执着自学完成的。正因为如此，要独立编写一本算法竞赛的书籍，对陈锋来说是一项巨大的挑战。令人高兴的是，他做到了，而且做得很棒！

本书的问世，更让我确信了两件事：

第一，“半路出家”的算法爱好者也可以通过自己的努力变得很出色，并不一定要从小接受严格的教育和训练。你看看书里的题目，有些可是顶级选手也不一定敢在比赛中挑战的。

第二，算法竞赛并不是脱离现实的“高级应试教育”。不然的话，一个看似完全不需要和算法打交道的软件工程师，干嘛要花费那么大的精力去学习算法、做题呢？

陈锋的成长轨迹很有代表性。从某种意义上说，他写的东西更能引起读者的共鸣。而且他为人热情、诚恳，可以比我有更多时间和精力与读者交流。事实上，他在写作期间已经与多位中学、大学选手和教师讨论了，目前负责维护丛书的 github 仓库和 wiki。如果你正在学习（或者刚学完）《算法竞赛入门经典（第 2 版）》，相信本书不会让你失望！

刘汝佳

前　　言

“请问《算法竞赛入门经典（第 2 版）》有没有配套题解啊？很多练习题好难，真希望能有一本简单、易懂的参考解答！”经常有读者追问类似的问题。笔者在进行训练学习时，也经常会有这样的想法。虽然很多题目可以在网上搜到对应题解，但这些题解多数是解题者为方便自己做题而随手记录的，解答过程未必严密、系统，语言表达上也比较随意，初学者理解起来就有一定的难度。

多年之前，笔者曾有幸参与了《算法竞赛入门经典——训练指南》一书的编写工作，收获颇大。也正是那次，我深刻感受到了自己在算法领域的不足，以及思维能力的亟待提升。私下里，我曾和刘汝佳老师商量，就以《算法竞赛入门经典（第 2 版）》的习题为训练题目，强迫自己在解出每道题之后，再对自己的思路进行严密、仔细的剖析，通过大量的训练，使自己得到一次系统的训练和提升。这次训练，使我记了厚厚一大本的笔记，而这本笔记就是本书的缘起。

希望本书能帮助更多跟我一样迫切需要提升算法思维能力的初学者！

算法有什么用

我大学学的是机械专业，但由于对数学非常热爱，加之毕业后发现软件行业貌似比较好“混”，且工资待遇比其他行业高些，所以就进入了开发领域。经过一段时间的工作后，我发现自己经常会遇到以下一些问题：

- 程序稍微复杂一些，代码就会写的很乱。
- 程序出了问题，不知道该如何调试，只会到处修改，然后再看效果。
- 用户需求稍作改变，就想骂街。
- 特别重要的一点是，如果你想跳到外企去工作，面试时肯定会让你编一些很难的算法程序。

后来，我进入到了微软上海全球技术支持中心做外包技术支持，接触到了许多严谨、求是、好学的工程师前辈。从他们身上，我学到了一些非常有效的解决问题的思路，以及那种“活到老学到老”的人生态度。

我逐渐明白：程序是要设计的。为了设计得清晰，需要学习数据结构、操作系统原理等非常多的基础知识，而这些体系本质上是前辈人思维方法的结晶。

另外，令很多程序员头疼的调试过程，给我印象最深的是一句话：调试的本质实际上就是在定位。大多数时候，调试的过程（并发程序的调试可能就更复杂些）其实就是一个二分查找：假如有 100 行程序结果不对，就可以在第 50 行看看结果是否符合预期，如果 OK，说明问题出在后 50 行，否则前 50 行一定有问题。如此递归下去，很快就能精准定位到有问题的代码。了解二分查找的朋友都知道，这个算法复杂度是 O(logn)。

用 C#开发服务端程序时，我经常会遇到内存问题，需要对垃圾收集（GC）的过程进行分析调试。深入学习之后我发现，其实 GC 模型的本质就是有向图。抱着这个思路再来分析解决内存问题，思路瞬间清晰了很多。

这样的例子还有很多。

在不断解决各类问题的过程中，我逐渐明白了——算法在本质上是诸多计算机学术以及实践领域积累下来的分析解决各种问题的思维方法。它不是象牙塔内的纯学术研究，更不是一堆仅能用来解决特定领域性能问题的高精尖技术。这个行业的技术人员，本质上正是以这些思维方法为武器，高效解决着不同行业领域不断涌现出的各类纷繁问题和挑战。

说到这里，我想到其他很多行业：京剧艺人每天早上要练嗓子，相声演员每天要练贯口，军人在战斗之余要进行大量训练，中医在繁忙之余要天天钻研《伤寒论》《黄帝内经》等经典……类似这样，需要认真对待并把基本功训练作为生活一部分的行业还有很多。对于笔者来说，算法思维就是 IT 相关行业的技术人员需要用同样态度持续不断进行训练的一项基本功。

所以，就有了这些年的学习过程，以及以本书作为省察的一个小小总结。

内容安排

本书内容分为以下 5 章。

第 1 章是各种编程训练技巧以及 C++11 语法特性的简单介绍。

第 2 章精选了一部分《算法竞赛入门经典（第 2 版）》的习题进行分析、解答，主要是读者反映较多的第 3~11 章的课后习题部分。

第 3 章是 ACM/ICPC 比赛真题分类选解，挑选了近些年 ACM/ICPC 比赛中较有价值的题目进行分析并解答。

第 4 章是比赛真题选译，整理并翻译了近几年来各大区域比赛中笔者认为值得学习训练的比赛真题。

第 5 章是比赛难题选译，内容类似于第 4 章，只是题目难度更上一个台阶。

关于 C++语言的使用

本书在解答各类算法题目时，使用 C++作为主要的编程语言，尽量使用 STL 中提供的现成数据结构，同时也尽量使用 C++11 的新特性。因为笔者认为，算法训练最关键的是训练解决问题的思维能力，包括抽象能力、分析能力、调试能力等，应该充分利用语言提供的语法特性使得程序更加简洁清晰，从而使解题者更专注于问题的抽象和分析本身。

关于题目代码

本书中的所有题目，笔者都是先完成代码并在线提交 AC（Accepted），然后才开始编写对应的分析题解。有些需要附上代码的题目，笔者会尽可能把代码的主要部分（去掉模板代码以及 C++的 namespace 导入部分）附在题目后面，但由于篇幅原因，实在无法全部



放入书中。还有些题目，虽然已经在线提交 AC，但由于无法严格证明题目的正确性，也没有在书中提供题解。

书中的所有代码，读者朋友们如有需要，可以通过如下网站进行下载：https://github.com/sukhoeing/aoapc-bac2nd-keys。

勘误和支持

虽然笔者已竭尽全力，力求减少纰漏，但由于水平有限，书中难免仍存在错漏之处，恳请广大读者朋友们批评指正。欢迎您将学习过程中遇到的各类问题、您对本书的想法以及宝贵意见，通过本书网站的 issues 部分一起交流。

致谢

首先要感谢刘汝佳老师，是他把我带进了算法艺术的大门，并且在工作极其繁忙的情况下一直耐心地指导着我的算法学习。

从小父亲就告诉我，对的事情一定要坚持。这句话支撑着我渡过了很多艰难的日子。同时，也要感谢我的太太梁明珠和女儿陈婉之。这三年来，我牺牲了大量本该陪伴他们的时间，投入到了本书的创作中。没有你们的支持和包容，我不可能完成这本书。

还要感谢微软工作期间经常指导我的老师张羿，从他那里，我第一次知道了世界上还有 ACM/ICPC 这回事。在如何做好一个程序员这件事上，他给了我非常多有价值的指导和帮助。

本书初稿完成之后，许多同学和朋友踊跃参与到了本书的试读中，并且提出了许多有价值的意见和反馈，他们的名字（排名不分先后）是陈飞、崔晨、杨恒杰、林永康、陈坤泽、孙博昊等。另外，书中的部分题目也参考了许多网友的在线题解，在此一并表示感谢。

最后要感谢清华大学出版社的贾小红编辑，用极大的耐心容忍着我把交稿时间一拖再拖，希望本书不会让您失望。

陈锋

目　　录



第 1 章　编程技巧与 C++11 语法特性介绍

对编程技巧和编程语言语法的熟练掌握有助于提高编码速度和准确率。本章介绍一些笔者在训练过程中总结出来的编程技巧和代码片段，以及 C++11 的语法新特性，希望能够对读者有所帮助。

1.1　编 程 技 巧

本节介绍一些在使用 C++语言进行代码编写以及调试时可能用到的技巧以及常见问题。

1.1.1　排序性能问题

相对于 C 语言内置的 qsort 函数，C++中提供的 sort 函数使用起来更加方便，不需要做指针类型转换。sort 有两种用法：第一种是传入一个 functor 对象，另外一种是直接传入一个排序函数，而笔者发现这两种用法语义上都是正确的，但是笔者实际测试发现使用 functor 的版本比直接使用函数的版本快不少，测试代码如下：

```
using namespace std;
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
const int N = 10000000;
struct TS{
   int a, b, c;
};
inline bool cmp (const TS& t1, const TS& t2) {
   if(t1.a != t2.a) return t1.a < t2.a;
   if(t1.b != t2.b) return t1.b < t2.b;
   return t1.c <= t2.c;
}

int cmp4qsort(const void * a, const void * b) {
   TS *t1 = (TS*)a, *t2 = (TS*)b;
   if(t1->a != t2->a) return t1->a - t2->a;
   if(t1->b != t2->b) return t1->b - t2->b;
   return t1->c - t2->c;
}
```

```
struct cmpFunctor {
  inline bool operator() (const TS& t1, const TS& t2) {
    if(t1.a != t2.a) return t1.a < t2.a;
    if(t1.b != t2.b) return t1.b < t2.b;
    return t1.c <= t2.c;
  }
};

TS tss[N];

void genData(){
    _for(i, 0, N) {
        tss[i].a = rand();
        tss[i].b = rand();
        tss[i].c = rand();
    }
}

int main()
{
    srand(time(NULL));

    genData();
    clock_t start = clock();
    sort(tss, tss+N, cmp);
    printf("sort by funtion pointer : %ld\n", clock() - start);

    genData();
    start = clock();
    sort(tss, tss+N, cmpFunctor());
    printf("sort by functor : %ld\n", clock() - start);

    genData();
    start = clock();
    qsort(tss, N, sizeof(TS), cmp4qsort);
    printf("qsort by funtion pointer : %ld\n", clock() - start);

    return 0;
}
/*
    g++ 4.8.0 result: 编译参数 -O2
    sort by funtion pointer : 36732
```

```
   sort by functor : 6324
   qsort by function : 15996
*/
```

笔者的机器上测试发现，STL 的 sort 使用 functor 的版本是最快的，比 qsort 都快一倍多。而使用 sort 传入函数指针的版本速度是最慢的，相对于前两者有大约 6 倍和 3 倍的差距，会在一些对排序性能要求很高的题目中形成比较明显的瓶颈，提醒读者注意。

1.1.2　整数输入

最经常输入的数据类型就是 int，经常需要输入之后直接插入到一个集合或者数组中，一般的做法是建立一个临时变量，使用 cin 或者 scanf 输入之后，再将这个临时变量插入到集合中。这样稍显烦琐。可以封装读取的函数并且这样调用：

```
int readint(){
    int x; scanf("%d", &x); return x; //此处 scanf 也可以根据需要换成 cin>>x
}
vector<int> vc;
vc.push_back(readint());
```

1.1.3　循环宏定义

算法比赛中，写得最多的代码就是像这样的循环代码：

```
for(int i = 0; i < N; i++) {}
```

这里 N 也可能是一个 STL 中集合的大小，如 vector.size 之类的。许多竞赛选手习惯使用大量的宏定义来简化代码，笔者最常用的宏定义是简化这个循环的：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
```

这样写循环时，就会简化成_for(i, 0, N)，这里的 a、b 两个参数都可传入表达式，例如：

```
vector b;
_for(i, 1, a.size()){...}
```

宏使用得当，可以大量简化代码，最典型的例子是本书习题 9-18 中有一个五维的 DP，里面有一个 5 层 for 循环，使用宏之后，可精简的代码非常可观。

另外一个比较有用的是：

```
#define _rep(i,a,b) for(int i=(a); i<=(b); ++i)
```

1.1.4　STL 容器内容调试输出

比赛中经常用到 STL 中的容器类，如 vector 和 set，而且在调试过程中经常需要输出这些容器的内容，每次都要写循环来输出，非常烦琐。笔者封装了两个泛型函数使用 C++的

IO 流对集合进行输出：

```
template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v) {
   for(int i = 0; i < v.size(); i++) os<<v[i]<<" ";
   return os;
}

template<typename T>
ostream& operator<<(ostream& os, const set<T>& v) {
   for(typename set<T>::iterator it = v.begin(); it != v.end(); it++)
os<<*it<<" ";
   return os;
}
```

使用方法如下：

```
vector<int> a; a.push_back(1); a.push_back(2); a.push_back(3);
cout<<a;                //输出 1 2 3

set<string> b; b.insert("1"); b.insert("2"); b.insert("3");
cout<<b;                //输出 1 2 3
```

1.1.5　二维几何运算类

在许多牵涉位置计算的题目（如本书习题 3-5）中，需要模拟物体位置并且进行移动和转向，如果每次都直接用 x 和 y 坐标分别计算，非常烦琐，其实可以使用《算法竞赛入门经典——训练指南》一书第 4 章中的几何操作类，复用向量的移动、旋转等逻辑，详细代码请参考相关章节。

```
struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
  Point& operator=(Point& p) : { x = p.x; y = p.y; return *this; }
};
typedef Point Vector;

Vector operator+ (const Vector& A, const Vector& B) { return Vector(A.x+B.x,
A.y+B.y); }
Vector operator- (const Point& A, const Point& B) { return Vector(A.x-B.x,
A.y-B.y); }
Vector operator* (const Vector& A, int p) { return Vector(A.x*p, A.y*p); }
bool operator== (const Point& a, const Point &b) { return a.x == b.x && a.y
== b.y; }
```

```
bool operator< (const Point& p1, const Point& p2) { return p1.x < p2.x ||
(p1.x == p2.x && p1.y < p2.y); }
istream& operator>>(istream& is, Point& p) { return is>>p.x>>p.y; }
```

1.1.6　内存池

在一些题目中，需要动态分配对象。例如，表达式解析时需要动态分配语法树的结点对象。一般的做法是直接用数组开辟空间，但是未必容易事先估计出需要开辟的空间大小，在逻辑控制中还要维护一个变量进行分配和释放，如果是多种对象都要动态分配，则更加烦琐。笔者基于 vector 容器和 C++的内存分配机制，编写了一个内存池：

```
template<typename T>
struct Pool {
    vector<T*> buf;
    T* createNew() {
        buf.push_back(new T());
        return buf.back();
    }

    void dispose() {
        for(int i = 0; i < buf.size(); i++) delete buf[i];
        buf.clear();
    }
};
```

使用方法如下：

```
struct Node{...};
struct Node2{...}

Pool <Node> n1Pool;
Pool <Node2> n2Pool;
//要分配内存构造新对象时：直接就是 Node *p = n1Pool.createNew();
Node2 *p2 = n2Pool.createNew();
```

然后在每次需要释放时直接调用 dispose 方法即可，不需要再维护各种中间变量。

1.1.7　泛型参数的使用

入门经典中很多算法的封装都会在某个结构体内部开一个数组，并且使用一个类似于 MAXSIZE 的结构来全局定义这个数组的大小，典型的如图论中的 Dijkstra 等算法：

```
const int MAXSIZE;
struct Dijkstra{
```

```
    int n, m, d[MAXSIZE], p[MAXSIZE];
...
}
```

如果同一个题目（如《算法竞赛入门经典——训练指南》中的习题 UVa10269 Adventure of Super Mario）中需要在两个不同的部分都用到 Dijkstra 算法怎么办？这个时候一般的做法就是定义多个 MAXSIZE 变量，但是会比较烦琐，也容易出错。

其实可以引入 C++的泛型参数来解决这个问题：

```
template<int MAXSIZE>
struct Dijkstra{
int n, m, d[MAXSIZE], p[MAXSIZE];
...
```

使用时，就可以通过下面的方式来指定不同的 MAXSIZE：

```
Dijkstra<MAXK * MAXN> sd;
Dijkstra<MAXN> pd;
```

具体使用可以参考训练指南中 UVa10269 的实现代码。

1.1.8 位运算操作封装

在使用位向量表示集合或进行状态压缩时，有个常用操作就是取得一个整数中某一位或者连续几位对应的 int 值。这些代码写起来较为烦琐，如果一个题目中多处调用，会增加出错的可能，笔者针对这种情况封装了一个位运算的操作类：

```
template<typename TI> //TI 可以是支持位操作的任何类型，一般是 int/long long
struct BitOp{
    //反转 pos 开始，长度为 len 的区域
    inline TI flip(TI op, size_t pos, size_t len = 1) { return op ^ (((1<<len)-1)
<< pos); }

    //取得从 pos 开始，长度为 len 的区域对应的整数值
    inline TI& set(TI& op, size_t pos, int v, size_t len = 1) {
        int o = ((1<<len)-1);
        return op = (op&(~(o << pos))) | ((v&o) << pos);
    }

    //取得从 pos 开始，长度为 len 的区域对应的整数值
    inline int get(TI op, size_t pos, size_t len = 1) { return (op >> pos)
& ((1<<len)-1); }

    //输出整数的二进制表示
    ostream& outBits(ostream& os, TI i) {
```



```
        if (i) outBits(os, (i >> 1)) << (i & 1);
        return os;
    }
};
```

如果是 32 位整数位运算可以使用 BitOp<long>来调用，64 位可以使用 BitOp<long long>来调用。

1.1.9　编译脚本

一般都是使用 g++编译然后在命令行运行，每次编译都要输入一堆命令，效率较低，所以笔者使用 Windows 命令行开发了两个脚本。

（1）编译脚本（ojc.bat）：这里假设 ojc.bat 以及 g++.exe 所在的目录已经加入到系统 PATH 环境变量中：

```
cls
g++ "%1" -lm -O2 -pipe -o"%~n1.exe"
```

使用方法如下：

```
ojc UVa100.cc
```

（2）编译并且直接运行（ojr.bat）：这里同样假设 ojr.bat 以及 g++.exe 所在的目录已经加入到系统 PATH 环境变量中。ojr.bat 的内容如下：

```
cls
echo 编译
del %~n1.exe
@g++ "%1" -lm -O2 -pipe -o"%~n1.exe"
@%~n1.exe<%~n1.in
```

以下命令会直接编译源文件，然后直接从 UVa100.in 读入数据运行：

```
ojr UVa100.cc
```

1.2　C++11 语言特性介绍

笔者写作本书时主流的算法比赛以及在线 OJ 平台均已支持最新的 C++11 语言标准。从开发者的角度来看，新标准中提供了不少能提高开发效率的新特性。本节选择了一些在算法比赛中常用特性进行介绍，希望读者通过练习掌握这些语言特性，提高在比赛中的编码速度和正确率。本书后文中的题解代码也有较多使用 C++11 的案例，请读者参考。

需要注意的是，如果是使用 g++编译，编译器需要加命令行参数-std=c++11。如果用 Visual Studio，则至少需要 2013 版本。在各大 OJ 在线提交时，也要选择 C++11。

1.2.1 类型推导（auto）

如果要使用比较长的类型声明，最常见的就是 STL 中的枚举器（iterator），就要写得很长，例如：

```
vector<int> vec;
vector<int>::iterator cit = vec.begin();
```

而在 C++11 中就可以这么写：

```
auto cit = vec.begin();
```

编译器遇到 auto 之后会根据右边的表达式自动推导出其具体类型。同时也支持引用类型的变量：

```
vector<int> vec = {1,2,3};
auto& v2 = vec[1];
v2 += 3;                //vec 就变成{1, 5, 3}
```

1.2.2 空指针值（nullptr）

在之前的 C/C++代码中，如果要表示空指针，一般使用“p = NULL;”，实际上 NULL 只是一个定义为常整数 0 的宏，这样有时候就可能和整数类型混淆。

在 C++11 中，有专门的用来表示空指针的数据类型：nullptr。nullptr 关键字代表值类型 std::nullptr_t，在语义上可以被理解为空指针。

之前的写法：

```
char *p = NULL;
int i = NULL;           //这里不会报错，因为 NULL 本质上就是 0
```

C++11 中的写法：

```
char *p = nullptr;
int i = nullptr;        //这里会报错，因为 nullptr 不再是整数类型
if(p)                   //这里仍然可以转换为 bool 的 false
```

1.2.3 容器的 for 循环遍历

以前去遍历一个 STL 中的集合（如 vector<int>）时要写出非常烦琐的代码：

```
vector<int> vec;
for(vector<int>::iterator it = vec.being(); it != vec.end(); it++){
    *it += 2;
    cout<<*it<<endl;
}
```



到了 C++11 中，其实可以这么写：

```
for(const auto& p : vec) {
cout<<p<<endl;
}
```

或者如果要修改容器中的数据：

```
for(auto& p : vec) p += 2;
```

这个语法和 Java 中的遍历方式非常像。其实不仅仅是 vector，所有的标准容器，如 map、string、deque、list，甚至数组都可以这么遍历，非常方便。

```
int arr[] = {1,2,3,4,5};
for(int& x : arr) x += 2;
```

1.2.4　匿名函数（Lambda）

匿名函数是笔者认为最重要的改进，是函数式编程（Funcitonal Programming Style）风格的基石。简单地说，就是可以在需要的地方定义函数，而不是提前定义好才能用：

```
using namespace std;
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
const int N = 10000000;
struct TS{
   int a, b, c;
};

void genData(){
   _for(i, 0, N) {
      tss[i].a = rand();
      tss[i].b = rand();
      tss[i].c = rand();
   }
}

int main()
{

   genData();
   sort(tss, tss+N, [](const TS& t1, const TS& t2) {
      if(t1.a != t2.a) return t1.a < t2.a;
      if(t1.b != t2.b) return t1.b < t2.b;
      return t1.c <= t2.c;
   });
```

```
    return 0;
}
```

以 C++98 的 STL 中 for_each(InputIterator first, InputIterator last, Function fn)为例，第 3 个参数需要一个 functor（函数对象）。所谓函数对象，其实是一个类，这个类重载了 operator，于是这个对象可以像函数一样被使用。

以前 STL 中的很多算法都是需要传入 functor 的，写起来非常麻烦。C++11 中，就可以直接用 lambda 代替。另外，利用 C++的 lambda 函数内部也可以对外围作用域的变量进行捕捉：

```
vector<int> list{1,2,3};
int total = 0;
for_each(list.begin(), list.end(), [&total](int x) { //匿名函数，捕捉 total
    total += x;
});
cout << total<<endl;
```

上述代码中的 lambda 函数内部要对 total 变量进行写操作，所以声明的[&total]部分对 total 进行按引用捕捉。

另外，还可以直接像声明一个变量一样声明一个函数：

```
//将 lambda 赋值给有类型的变量然后作为参数传递
total = 0;
std:function<void(int)> add = [&total](int x) { total += x;   };
for_each(begin(list), end(list), add);
cout << total<<endl;
```

或者声明的类型部分也可以直接使用类型推导：

```
total = 0;
auto add2 = [&total](int x) { total += x; }; //类型推导 lambda 的类型
for_each(begin(list), end(list), add2);
cout << total<<endl;
```

关于 lambda 的用法，有非常大的想象空间。建议读者参考以下资料仔细学习：https://msdn.microsoft.com/zh-cn/library/dd293608.aspx。

1.2.5 统一的初始化语法

在 C++98 中，对于数组可以这样初始化其内容：

```
int arr[] = {1,2,3};
```

但是对于 STL 中的容器，就必须一个一个元素进行附加：

```
vector<int> vec;
vec.push_back(1); vec.push_back(2); vec.push_back(3);
```

在 C++11 中，可以使用像数组那样的初始化语法对 STL 容器进行初始化：

```
vector<string> vec{1,2,3};
map<string, string> dict{ {"ABC", "123"}, {"BCD", "234"}}; //map 也可以
```

1.2.6　哈希容器

比赛中，经常有用哈希容器存储数据的需要，而 C++98 标准的 STL 中并没有提供基于 hash 算法的容器，基于平衡二叉树实现的 map 可以起到类似的作用，但是在数据量较大时速度还是不够快（查询时间复杂度是 O(logn)的），有时就不得不自己手动编写 Hash 算法。而在 C++11 中正式引入了几个基于 Hash 算法的容器：unordered_map、unordered_set、unordered_multimap 和 unordered_multiset。

当不需要元素排序时，可以尽量使用这些容器来获得更好的查找性能。

```
unordered_map<string,int> um {
   {"Dijkstra",1972}, {"Scott",1976},
   {"Wilkes",1967}, {"Hamming",1968}
};
um["Ritchie"] = 1983;
for(auto x : um) cout << '{' << x.first << ',' << x.second << '}';
```

其他 Hash 容器的用法类似。

默认的 Hash 容器只是提供了内置数据类型的 Hash 算法，如果是自定义类型，就需要提供自定义的 Hash 函数。自定义类型可能包含几种内置类型，可以分别算出其 Hash，然后对它们进行组合得到一个新的 Hash 值，一般直接采用移位加异或（XOR）便可得到基本够用的哈希值（碰撞不太频繁）。容器处理碰撞时需判断两对象是否相等，所以必须提供判断相等的方法，建议重载“==”操作符：

```
#include <unordered_map>
#include <string>
#include <iostream>

using namespace std;

struct Type
{
   int x; string y;
   bool operator==(const Type& a) const {
      return x == a.x && y == a.y;
   }
```

```
};

struct HashFunc
{
    std::size_t operator()(const Type &o) const
    {
     return ((hash<int>()(o.x)
            ^ (hash<string>()(o.y) << 1)) >> 1);
    }
};

int main(){
    unordered_map<Type, string, HashFunc> testHash =
    {
        { { 1, "1"}, "one" },
        { { 2, "2"}, "two" },
        { { 3, "3"}, "three"}
    };

    for(const auto& kv : testHash)
         cout<<kv.first.x<<","<<kv.first.y<<" - "<<kv.second<<endl;
return 0;
}
/*
```

输出:

```
3,3 - three
2,2 - two
1,1 - one
*/
```

第 2 章　《算法竞赛入门经典（第 2 版）》习题选解

2.1　数组和字符串

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 3 章。

习题 3-1　得分（Score, ACM/ICPC Seoul 2005, UVa1585）

给出一个由 O 和 X 组成的串（长度为 1～80），统计每个字符的得分之和。每个 O 的得分为已经连续出现的 O 的个数，X 得分为 0。例如，OOXXOXXOOO 的得分为 1+2+0+0+1+0+0+1+2+3。

【分析】

使用 for 循环对输入串的字符进行遍历，维护一个已经连续出现的'O'个数的计数器 cnt 以及串的得分和 sum。初始 cnt = 0，sum = 0。如果遇到'O'就++cnt，然后把 cnt 加到 sum 中，如果遇到'X'就重置 cnt 为 0。

完整程序（C++11）如下：

```
int main() {
    int T;
    char buf[128];
    scanf("%d\n", &T);
    while(T--) {
        gets(buf);
        int cnt = 0, sum = 0, sz = strlen(buf);
        _for(i, 0, sz){
            if(buf[i] == 'O') sum += (++cnt);
            else cnt = 0;
        }
        printf("%d\n", sum);
    }
    return 0;
}
```

习题 3-2　分子量（Molar Mass, ACM/ICPC Seoul 2007, UVa1586）

给出一种物质的分子式（不带括号），求分子量。本题中的分子式只包含 4 种原子，分别为 C、H、O、N，原子量分别为 12.01、1.008、16.00、14.01（单位：g/mol）。例如，C_6H_5OH 的分子量为 6× (12.01 g/mol) + 6 × (1.008 g/mol) + 1 × (16.00 g/mol)=94.108g/mol。

【分析】

依次扫描即可，注意原子后面不带数目的情况。扫描的过程中，维护一个当前已经输

入的数字字符组成的数字 cnt。一开始以及遇到一个新原子时，cnt=−1，表示“还未开始计数”的状态。方便遇到原子后不带数目以及数字有多位的情况处理。

和习题 3-1 类似，也要在循环结束之后处理最后一个原子。完整程序如下：

```
int main(){
   int T, cnt, sz;
   double W[256], ans;
   char buf[256], c, s;
   W['C'] = 12.01, W['H'] = 1.008, W['O'] = 16.0, W['N'] = 14.01;
   scanf("%d\n", &T);
   while(T--){
      scanf("%s", buf);
      ans = 0;
      s = 0; cnt = -1; sz = strlen(buf);
      _for(i, 0, sz){
         char c = buf[i];
         if(isupper(c)) {
            if(i) {
               if(cnt == -1) cnt = 1;
               ans += W[s] * cnt;
            }
            s = c;
            cnt = -1;
         } else {
            assert(isdigit(c));
            if(cnt == -1) cnt = 0;
            cnt = cnt*10 + c - '0';
         }
      }
      if(cnt == -1) cnt = 1;
      ans += W[s] * cnt;
      printf("%.3lf\n", ans);
   }

   return 0;
}
```

习题 3-3　数数字（Digit Counting , ACM/ICPC Danang 2007, UVa1225）

把前 n（$n \leqslant 10000$）个整数按顺序写在一起：123456789101112…数一数 0～9 各出现多少次（输出 10 个整数，分别是 0, 1, …, 9 出现的次数）。

【分析】

因为 n 的最大值比较小，可以用建表的方式来计算。令 C[n][k]表示前 n 个数字写在一



起，k（$k=0$～9）总共出现几次，则有 C[n+1][k] = C[n][k] + x，其中 x 是 k 在 n 中出现的次数。直接按照这个公式就可以把所有答案提前计算出来，然后每读入一个 n 就直接输出预处理的结果即可。

习题 3-4　周期串（Periodic Strings, UVa455）

如果一个字符串可以由某个长度为k的字符串重复多次得到，就可以说该串以k为周期。例如，abcabcabcabc 以 3 为周期（注意，它也以 6 和 12 为周期）。

输入一个长度不超过 80 的字符串，输出它的最小周期。

【分析】

字符串的周期 p 只可能是闭区间[1,k]内能被 k 整除的数，然后从小到大遍历所有的 p，看看对于每个 i=0～k−1 是否符合 S[i] = S[i%C]，找到第一个全部符合的 p 就是所求结果。完整程序如下：

```
int main(){
    int N; scanf("%d", &N);
    char line[256];
    bool first = true;
    while(N--) {
        if(first) first = false;
        else puts("");
        scanf("%s", line);
        int sz = strlen(line);
        _rep(p, 1, sz){
            if(sz % p) continue;
            bool ans = true;
            _for(i, 0, p) {
                for(int j = i + p; j < sz; j+=p) {
                    if(line[j] != line[i]) { ans = false; break; }
                }
                if(!ans) break;
            }
            if(ans) { printf("%d\n", p); break; }
        }
    }

    return 0;
}
```

习题 3-5　谜题（Puzzle, ACM/ICPC World Finals 1993, UVa227）

有一个 5*5 的网格，其中恰好有一个格子是空的，其他格子各有一个字母。一共有 4 种指令：A、B、L、R，分别表示把空格上/下/左/右的相邻字母移到空格中。输入初始网格

和指令序列（以数字 0 结束），输出指令执行完毕后的网格。如果有非法指令，应输出“This puzzle has no final configuration.”，例如图 2.1 执行 ARRBBL0 后为图 2.2。

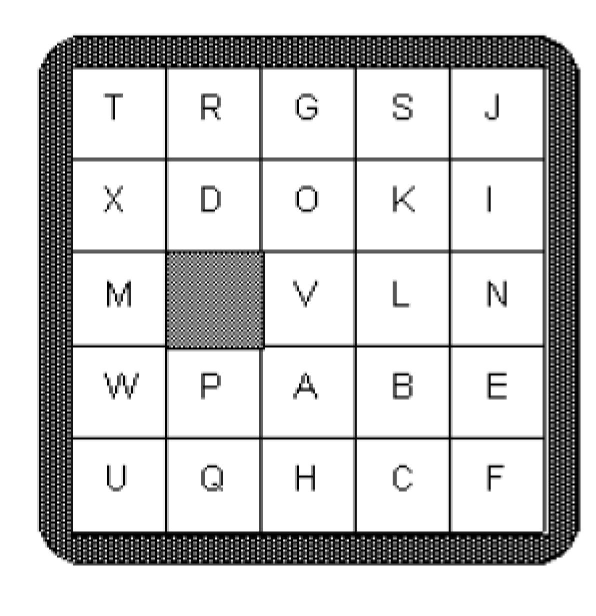


图 2.1

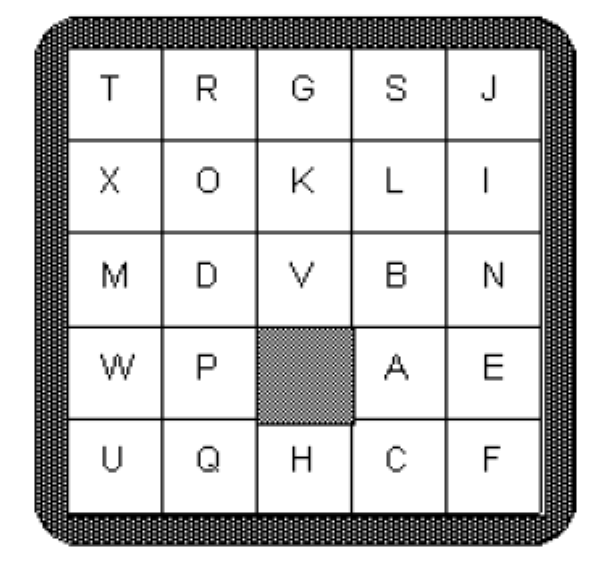


图 2.2

【分析】

使用以下结构表示坐标和向量：

```
struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
};
typedef Point Vector;
```

然后直接模拟即可，可以将 4 种指令字符以及对应 4 个方向的向量存放到一个 map<char, Vector>中，方便移动时计算新的空格位置。

完整程序如下：

```
using namespace std;

const int GSize = 5;
vector<string> grid;
Point ePos;
map<char, Vector> DIRS;

bool valid(const Point& p) {
    return p.x >= 0 && p.x < GSize && p.y >= 0 && p.y < GSize;
}

void printGrid() {
    for(int i = 0; i < GSize; i++) {
        for(int j = 0; j < GSize; j++) {
            if(j) cout<<' ';
            cout<<grid[i][j];
        }
        cout<<endl;
```

```
    }
}

bool tryMove(char cmd) {
    //cout<<"move "<<cmd<<":"<<endl;
    if(!DIRS.count(cmd)) return false;
    assert(DIRS.count(cmd));
    Point p = ePos + DIRS[cmd];
    if(!valid(p)) return false;
    swap(grid[p.x][p.y], grid[ePos.x][ePos.y]);
    ePos = p;

    //printGrid();
    return true;
}

int main()
{
    int t = 1;
    string line;
    DIRS['A'] = Vector(-1, 0); DIRS['B'] = Vector(1, 0); DIRS['L'] = Vector(0,
-1); DIRS['R'] = Vector(0, 1);
    while(true) {
        grid.clear();
        ePos.x = -1; ePos.y = -1;
        for(int i = 0; i < GSize; i++)
        {
            getline(cin, line);
            if(line == "Z") return 0;
            assert(line.size() == GSize);
            for(int j = 0; j < GSize; j++)
                if(line[j] == ' ') {
                    assert(ePos.x == -1 && ePos.y == -1);
                    ePos.x = i;
                    ePos.y = j;
                }
            grid.push_back(line);
        }
        char move;
        string moves;
        while(true) {
            getline(cin, line);
```

```
            assert(!line.empty());
            bool end = *(line.rbegin()) == '0';
            if(!end) moves.append(line);
            else moves.append(line, 0, line.size() - 1);
            if(end) break;
        }
        bool legal = true;
        for(int i = 0; i < moves.size(); i++)
            if(!tryMove(moves[i])) { legal = false; break; }

        if(t > 1) cout<<endl;
        cout<<"Puzzle #"<<t++<<":"<<endl;
        if(legal) printGrid();
        else cout<<"This puzzle has no final configuration."<<endl;
    }
    return 0;
}
```

习题 3-6 纵横字谜的答案（Crossword Answers, ACM/ICPC World Finals 1994, UVa232）

输入一个 r 行 c 列（$1 \leqslant r, c \leqslant 10$）的网格，黑格用“*”表示，每个白格都填有一个字母。如果一个白格的左边相邻位置或者上边相邻位置没有白格（可能是黑格，也可能出了网格边界），则称这个白格是一个起始格。

首先把所有起始格按照从上到下、从左到右的顺序编号为 1、2、3、…，如图 2.3 所示。接下来要找出所有横向单词（Across），这些单词必须从一个起始格开始，向右延伸到一个黑格的左边或者整个网格的最右列。最后找出所有竖向单词（Down），这些单词必须从一个起始格开始，向下延伸到一个黑格的上边或者整个网格的最下行。输入输出格式和样例请参考原题。

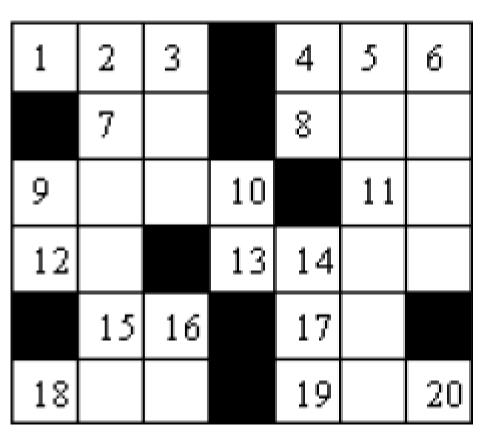


图 2.3

【分析】

还是和习题 3-5 一样，建立坐标和向量的结构方便进行位置的移动处理。依次扫描，用一个全局的 Point 数组 eligible 存放所有单词的起始点坐标，同时要用两个 vector<int>，即 across 和 down 来分别记录扫描出的横向单词和竖向单词的起点坐标在 eligible 中的编号。

在读取两个方向单词时，使用两个向量计算来读取所有的单词，即 dRight(0, 1)和 dDown(1, 0)，这样可以降低代码复杂度。具体请参见代码，完整程序如下：

```
using namespace std;

struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
};
```



```
typedef Point Vector;
Vector operator+ (const Vector& A, const Vector& B) { return Vector(A.x+B.x, A.y+B.y); }
int R, C;
const int MAXC = 16;
char grid[MAXC][MAXC];

inline bool valid(const Point& p) { return p.x >= 0 && p.x < R && p.y >= 0 && p.y < C; }
int main(){
   char buf[MAXC]; int bufLen;
   const Vector dLeft(0, -1), dUp(-1, 0), dRight(0, 1), dDown(1, 0);
   for(int t = 1; scanf("%d%d", &R, &C) == 2 && R; t++) {
      vector<Point> eligible;
      vector<int> down, across;

      if(t > 1) puts("");
      printf("puzzle #%d:\n", t);
      _for(i, 0, R){
         scanf("%s", grid[i]);
         _for(j, 0, C){
            if(grid[i][j] == '*') continue;
            Point p(i, j), left = p + dLeft, up = p + dUp;
            bool isCross = !valid(left) || grid[left.x][left.y] == '*';
            bool isDown = !valid(up) || grid[up.x][up.y] == '*';
            if(isCross) across.push_back(eligible.size());
            if(isDown) down.push_back(eligible.size());
            if(isCross || isDown) eligible.push_back(p);
         }
      }

      puts("Across");
      for(auto n : across){
         bufLen = 0, memset(buf, 0, sizeof(buf));
         Point p = eligible[n];
         while(valid(p) && grid[p.x][p.y] != '*') {
            buf[bufLen++] = grid[p.x][p.y];
            p = p + dRight;
         }
         printf("%3d.%s\n", n+1, buf);
      }
      puts("Down");
```

```
        for(auto n : down){
            bufLen = 0, memset(buf, 0, sizeof(buf));
            Point p = eligible[n];
            while(valid(p) && grid[p.x][p.y] != '*') {
                buf[bufLen++] = grid[p.x][p.y];
                p = p + dDown;
            }
            printf("%3d.%s\n", n+1, buf);
        }
    }
    return 0;
}
```

习题 3-7　DNA 序列（DNA Consensus String, ACM/ICPC Seoul 2006, UVa1368）

输入 *m* 个长度均为 *n* 的 DNA 序列，求一个 DNA 序列，到所有序列的总 Hamming 距离尽量小。两个等长字符串的 Hamming 距离等于字符不同的位置个数，如 ACGT 和 GCGA 的 Hamming 距离为 2（左数第 1、4 个字符不同）。

输入整数 *m* 和 *n*（$4 \leqslant m \leqslant 50$，$4 \leqslant n \leqslant 1000$），以及 *m* 个长度为 *n* 的 DNA 序列（只包含字母 A、C、G、T），输出到 *m* 个序列的 Hamming 距离和最小的 DNA 序列和对应的距离。如有多解，要求字典序最小的解。例如，对于下面 5 个 DNA 序列，最优解为 TAAGATAC。

TATGATAC
TAAGCTAC
AAAGATCC
TGAGATAC
TAAGATGT

【分析】

对于所求结果序列 S 来说，Hamming 距离和最小意味着 S 中每一列的字符都在 *m* 个序列的对应列上出现次数最多。可以依次对 *m* 个序列中每一列的字符进行统计，字典序最小并且出现次数最多的那个就是 S 中这一列的字符。完整程序如下：

```
using namespace std;

struct ChCnt {
    int cnt;            //出现次数
    char c;             //字符
    void init(char ch = 'A'){
        c = ch;
        cnt = 0;
    }

    ChCnt() { init(); }
```

```
    bool operator<(const ChCnt& cc2) const {
        return cnt > cc2.cnt || (cnt == cc2.cnt && c < cc2.c);
    }
};

int main(){
    int T = 1, m, n;
    cin>>T;
    string line;
    vector<string> seqs;
    char IDX[256] = {0};
    IDX['A'] = 0; IDX['C'] = 1; IDX['G'] = 2; IDX['T'] = 3;
    while(T--) {
        seqs.clear();
        cin>>m>>n;
        for(int i = 0; i < m; i++) {
            cin>>line;
            assert(line.size() == n);
            seqs.push_back(line);
        }
        string ansStr; int ans = 0;
        vector<ChCnt> ccs(4);
        for(int i = 0; i < n; i++) {
            ccs[0].init('A');
            ccs[1].init('C');
            ccs[2].init('G');
            ccs[3].init('T');
            for(int j = 0; j < m; j++)
                ccs[IDX[seqs[j][i]]].cnt++;
            sort(ccs.begin(), ccs.end());    //先按照出现次数再按字符进行排序
            ansStr += ccs.front().c;
            ans += (m - ccs.front().cnt);
        }

        cout<<ansStr<<endl<<ans<<endl;
    }
    return 0;
}
```

习题 3-8　循环小数（Repeating Decimals, ACM/ICPC World Finals 1990, UVa202）

输入整数 a 和 b（$0 \leqslant a \leqslant 3000$，$1 \leqslant b \leqslant 3000$），输出 a/b 的循环小数表示以及其循环节长度。例如 a=5，b=43，小数表示为 0.(116279069767441860465)，循环节长度为 21。

【分析】

首先简单介绍一下长除法，以 3/7 为例，如图 2.4 所示。

```
       .4 2 8 5 7 1 4 ...
 7 ) 3.0 0 0 0 0 0 0 0
      2 8                                  30/7 = 4 r 2
        2 0
        1 4                                20/7 = 2 r 6
          6 0
          5 6                              60/7 = 8 r 4
            4 0
            3 5                            40/7 = 5 r 5
              5 0
              4 9                          50/7 = 7 r 1
                1 0
                  7                        10/7 = 1 r 3
                  3 0
                  2 8   这里开始循环       30/7 = 4 r 2
                    2 0
```

图 2.4

本题实际上就是模拟长除法的计算过程，其中每一次除法时都有被除数和余数，当被除数出现重复时就表示出现循环节了。所以需要记录每一次的被除数及其在循环小数中的位置，需要注意当除数不够除，每一次补零也需要记录其对应的位置。

完整程序如下：

```
using namespace std;

const int MAXN = 3000 + 5;
map<int,int> Pos;
void solve(int n, const int d, string& ans, int& r) {
   assert(n%d && n<d);
   ans = ".";
   Pos.clear();

   while(true) {
      n *= 10;
      int p = Pos[n];
      if(p == 0) Pos[n] = ans.size();
      else{
         r = ans.size() - p;              //找到循环节
         if(r > 50) { ans.erase(p + 50); ans += "..."; }
         ans.insert(p, "(");
         ans += ')';
```



```
            break;
        }

        if(n < d) { ans += '0'; continue; }          //补 0

        int div = n/d, mod = n%d;
        ans += (char)(div + '0');
        n = mod;
        if(n == 0) { ans += "(0)"; r = 1; break; }
    }
}

int main(){
    int a, b;
    while(scanf("%d%d", &a, &b) == 2) {
        string ans = ".(0)";
        int r = 1;                                   //循环节长度
        if(a%b) solve(a%b, b, ans, r);
        printf("%d/%d = %d%s\n", a, b, a/b, ans.c_str());
        printf("   %d = number of digits in repeating cycle\n\n", r);
    }

    return 0;
}
```

习题 3-9　子序列（All in All, UVa10340）

输入两个字符串 s 和 t，判断是否可以从 t 中删除 0 个或多个字符（其他字符顺序不变），得到字符串 s。例如，abcde 可以得到 bce，但无法得到 dc。

【分析】

可以使用两个变量 *i* 和 *j* 对两个字符串 s 和 t 同时进行遍历，对于每个 *i*，如果 t[*j*] != s[*i*]，那么一直对 *j* 进行递增操作，如果 *j* 越界，说明 t[*j*…]中不存在等于 s[*i*]的字符，查找失败。如果对于每个 *i* 匹配成功，则说明问题有解，否则无法从 t 中删除字符得到 s。完整程序如下：

```
const int LEN = 100024;
char s[LEN], t[LEN];
int main() {
    while (scanf("%s%s", s, t) == 2) {
        int sLen = strlen(s), tLen = strlen(t);
        bool ok = true;
        for (int i = 0, j = 0; i < sLen; i++, j++) {
```

```
            while (j < tLen && t[j] != s[i]) j++;
            if (j == tLen) { ok = false; break; }
        }
        printf("%s\n", ok ? "Yes" : "No" );
    }
    return 0;
}
```

习题 3-10　盒子（Box, ACM/ICPC NEERC 2004, UVa1587）

给定 6 个矩形的长和宽 w_i 和 h_i（$1 \leqslant w_i, h_i \leqslant 1000$），判断它们能否构成长方体的 6 个面。

【分析】

注意长方体的 6 个面一定是可以形成 3 对相同的矩形，并且边的长度刚好只有 3 个。对输入的矩形的长宽进行处理，使得长 $x \geqslant$ 宽 y。输入完按照先 x 后 y 对输入的矩形进行排序。排序完成之后，如果输入合法，应该就是 3 对矩形，每一对都应该完全一致；否则非法。

记排完序的矩形为 rects[6]，则长方体的 3 条边就是 rects[0].x、rects[4].x、rects[5].y，此时就可以按照这 3 个边长，把 6 个矩形重新构造出来，与输入数据比对。如果相同，说明合法，否则非法。

习题 3-11　换低档装置（Kickdown, ACM/ICPC NEERC 2006, UVa1588）

给出两个长度分别为 n_1、n_2（$n_1, n_2 \leqslant 100$）且每列高度只为 1 或 2 的长条，需要将它们放入一个高度为 3 的容器（如图 2.5 所示），问能够容纳它们的最短容器长度。

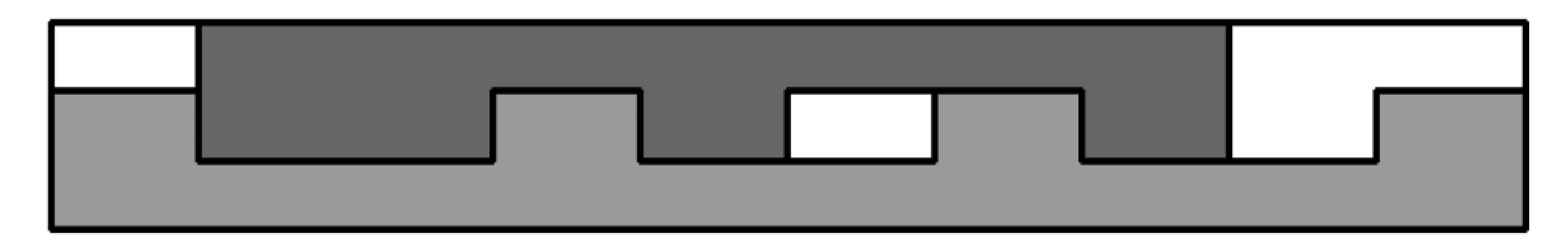


图 2.5

【分析】

数据范围比较小，可以直接遍历上方长条的所有位置，看看能不能和下方匹配即可。可以引入一维坐标，其中下方的长条起始点放在 100，依次遍历上方长条所有可能的起始点 $b2$，$b2$ 的可能范围是$[100-n_1, 100+n_1+n_2]$。对于一个起始点 $b2$，依次尝试放置序列的每一个点，看看数轴上对应点的两个长条对应列的高度和是否大于 3，如果大于 3，说明放不到容器里面；否则继续。这样用 $O(n_1 * n_2)$的时间复杂度就可以求出最短的容器长度。

习题 3-12　浮点数（Floating-Point Numbers, UVa11809）

计算机常用阶码-尾数的方法保存浮点数。如图 2.6 所示，如果阶码有 6 位，尾数有 8 位，可以表达的最大浮点数为 $0.111111111_2 \times 2^{111111_2}$。注意小数点后第一位必须为 1，所以一共有 9 位小数。



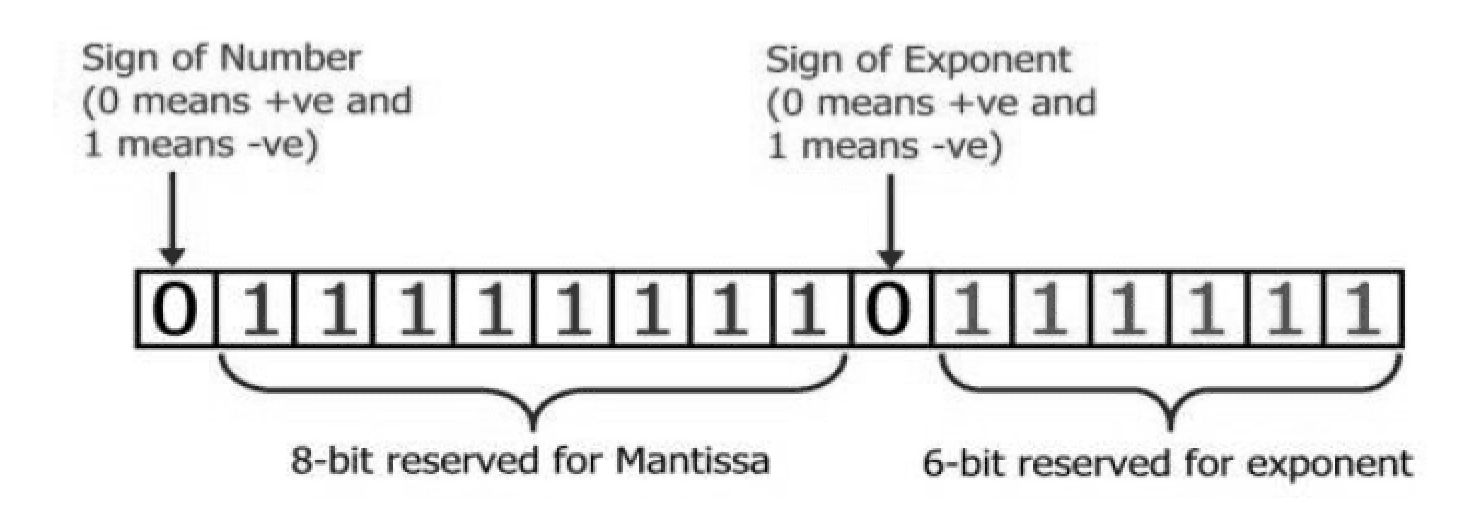


图 2.6

这个数换算成十进制之后就是 $0.998046875*2^{63}=9.205357638345294*10^{18}$。你的任务是根据这个最大浮点数，求出阶码的位数 E 和尾数的位数 M。输入格式为 AeB，表示最大浮点数为 $A*10^B$，$0<A<10$，并且恰好包含 15 位有效数字。输入结束标志为 0e0。对于每组数据，输出 M 和 E。输入保证有唯一解，且 $0\leqslant M\leqslant 9$，$1\leqslant E\leqslant 30$。在本题中，$M+E+2$ 不必为 8 的整数倍。

【分析】

根据题意可以推算出最大值 $v=\left(1-\dfrac{1}{2^{M+1}}\right)*2^{2^E-1}=A*10^B$。因为两边都比较大，所以可以同时求以 10 为底的对数：$\lg v=\lg(2^{M+1}-1)-(M+1)*\lg 2+(2^E-1)*\lg 2=\lg A+B$。

可以遍历所有可能的 M，根据上述公式求出 E 的值，然后再用 E 和 M 求出 $\lg v$ 和输入的值进行比较，如果相等，说明 M、E 就是所求的值。做两个浮点数相等判断时，二者之差的绝对值如果小于 1e−6，则认为二者相等。完整程序（C++11）如下：

```
const double EPS = 1e-6;
int main(){
    char line[256];
    double lg2 = log10(2), A, v; int B;
    while(scanf("%s", line) == 1 && strcmp(line, "0e0") != 0){
        *strchr(line, 'e') = ' ';
        sscanf(line, "%lf%d", &A, &B);
        v = log10(A)+B;
        _rep(M, 1, 10){
            int E = round(log10((v+M*lg2-log10(pow(2,M)-1))/lg2 + 1) / lg2);
            if(fabs(((1<<E)-1)*lg2 + log10(pow(2,M)-1) - M*lg2 - v) <= EPS) {
                printf("%d %d\n",M-1, E);
                break;
            }
        }
    }

    return 0;
}
```

注意，本题代码使用了 C++11 中提供的 round 函数（属于 C 语言的 math 标准库）来做四舍五入操作，不再需要使用类似 floor(x+0.5)这样的技巧。

浮点数在计算机科学中是非常重要的课题，有兴趣的读者可以参考如下链接：http://share.onlinesjtu.com/mod/tab/view.php?id=176。

2.2　函数和递归

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 4 章。

习题 4-1　象棋（Xiangqi, ACM/ICPC Fuzhou 2011, UVa1589）

考虑一个象棋残局，其中红方有 n（$2 \leqslant n \leqslant 7$）个棋子，黑方只有一个将。红方除了有一个帅（G）之外还有 3 种可能的棋子：车（R）、马（H）、炮（C），并且需要考虑蹩马腿（如图 2.7 所示）和将与帅不能照面（将帅如果同在一条直线上，中间又不隔着任何棋子的情况下，走子的一方获胜）的规则。

输入所有棋子的位置，保证局面合法并且红方已经将军。你的任务是判断红方是否已经把黑方将死。关于中国象棋的相关规则请参见原题。

【分析】

要判断黑方是否必死，其实就是反过来判断黑方是否有种走法，在走出一步之后能不被红方的任何一个棋子将死。首先判断，黑方是不是可以直接将红方将死，如果可以，就无须进行下一步的判断。

然后挨个尝试黑方的各种合法走法（水平或者垂直，但是不能走出黑子的大本营）。如果所有走法都会导致被红方某个棋子吃掉，说明红方必胜。

需要特别注意的是，黑方走子时是可以吃掉红方棋子的，如果有这种情况，需在吃子之后再判断输赢。

从实现过程中来说，有一个公共的过程可以抽取：就是判断一个棋子是否可以从一个点 p1 直接水平或者垂直地走到另外一个点 p2，中间有 0 个（车要吃子或者黑将直接将军）或者恰好 1 个棋子（红炮要将军）。实现中，需要将跳马的 8 个方向封装成向量。

习题 4-2　正方形（Squares, ACM/ICPC World Finals 1990, UVa201）

有 n 行 n 列（$2 \leqslant n \leqslant 9$）的小黑点，还有 m 条线段连接其中的一些黑点。统计这些线段连成了多少个正方形（每种边长分别统计）。

行从上到下编号为 1～n，列从左到右编号为 1～n。边用 H i j 和 V i j 表示，分别代表边(i,j)-(i,j+1)和(i,j)-(i+1,j)。例如图 2.8 最左边的线段用 V 1 1 表示。图 2.8 中包含 2 个边长为 1 的正方形和 1 个边长为 2 的正方形。

【分析】

对于每一个点(i,j)，记录一个向右延伸和向下延伸的最长线段长度 hExp 和 vExp，则二者都可以根据这个点上出发的线段类型来递推：



（1）如果存在向右的线段，则“hExp(i, j) = hExp(i, j+1) + 1;”。

（2）如果存在向下的线段，则“vExp(i,j) = vExp(i+1, j) + 1;”。

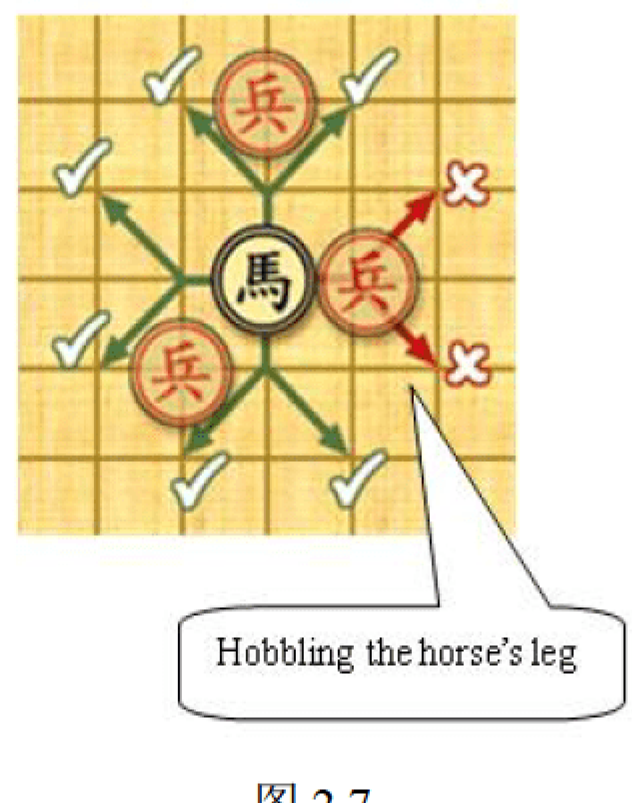


图 2.7

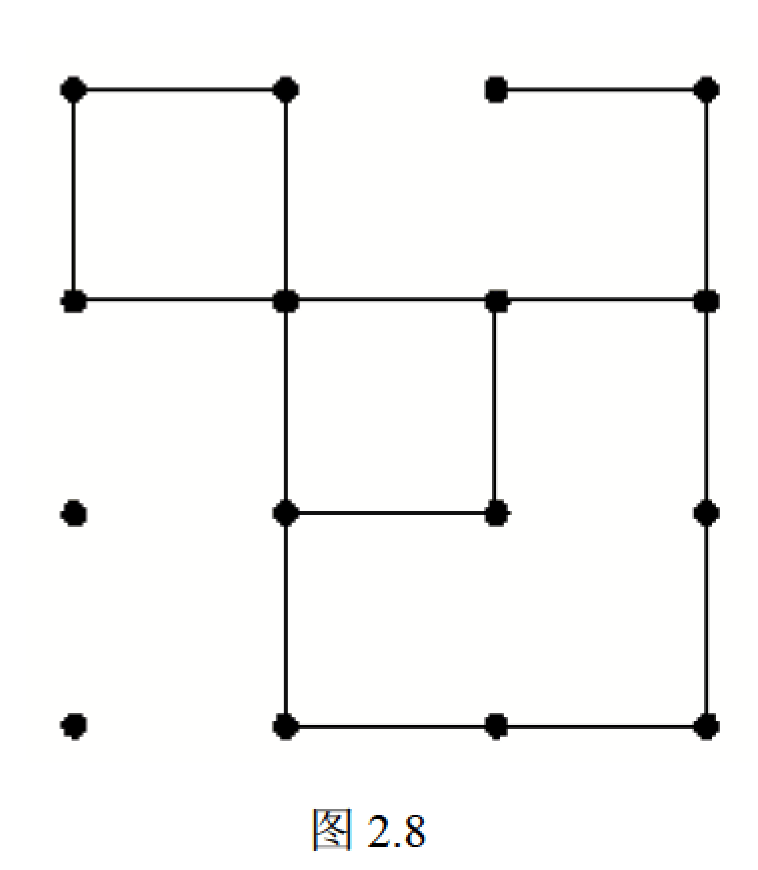


图 2.8

顺序遍历每一个点 P(i,j)，依次判断以 P 为左上顶点的可能正方形的边长 *s*，其中 *s* 为 1 到 min(hExp(i, j), vExp(i+1, j))的整数。然后可以根据 *s* 计算出正方形的左下以及右上顶点，再根据这两个点对应的 hExp 和 vExp 是否大于等于 *s*，即可判断能否形成长度为 *s* 的正方形。完整程序如下：

```
    using namespace std;
    const int MAXN = 16;
    int n, m, vExp[MAXN][MAXN], hExp[MAXN][MAXN], H[MAXN][MAXN], V[MAXN][MAXN],
Squares[MAXN];
    int main(){
        char buf[4]; int x, y;
        for(int t = 1; scanf("%d", &n) == 1; t++){
            if(t > 1) printf("\n**********************************\n\n");
            memset(vExp, 0, sizeof(vExp)), memset(hExp, 0, sizeof(hExp));
            memset(H, 0, sizeof(H)), memset(V, 0, sizeof(V)), memset(Squares, 0,
sizeof(Squares));

            scanf("%d", &m);
            _for(i, 0, m){
                scanf("%s%d%d", buf, &x, &y);
                if(buf[0] == 'H')  H[x][y] = 1; else V[y][x] = 1;
            }

            for(int i = n; i >= 1; i--) for(int j = n; j >= 1; j--) {
                if(H[i][j]) hExp[i][j] = hExp[i][j+1] + 1;
                if(V[i][j]) vExp[i][j] = vExp[i+1][j] + 1;
            }
```

```
        _rep(i, 1, n) _rep(j, 1, n) {
            int maxS = min(hExp[i][j], vExp[i][j]);
            _rep(s, 1, maxS) if(hExp[i+s][j] >= s && vExp[i][j+s] >= s)
Squares[s]++;
        }

        printf("Problem #%d\n\n", t);
        bool found = false;
        _rep(i, 1, n) if(Squares[i]) {
            found = true;
            printf("%d square (s) of size %d\n", Squares[i], i);
        }
        if(!found) puts("No completed squares can be found.");
    }
    return 0;
}
```

习题 4-3 黑白棋（Othello, ACM/ICPC World Finals 1992, UVa220）

你的任务是模拟黑白棋游戏的进程。黑白棋的规则为：黑白双方轮流放棋子，每次必须让新放的棋子“夹住”至少一枚对方棋子，然后把所有被新放棋子“夹住”的对方棋子替换成己方棋子。一段连续（横、竖或者斜向）的同色棋子被“夹住”的条件是两端都是对方棋子（不能是空位）。图 2.9 中的白棋有 6 个合法操作，分别为(2,3),(3,3),(3,5),(6,2),(7,3),(7,4)。选择在(7,3)放白棋后变成图 2.10（注意有竖向和斜向的共两枚黑棋变白）。注意(4,6)的黑色棋子虽然被夹住，但不是被新放的棋子夹住，因此不变白。

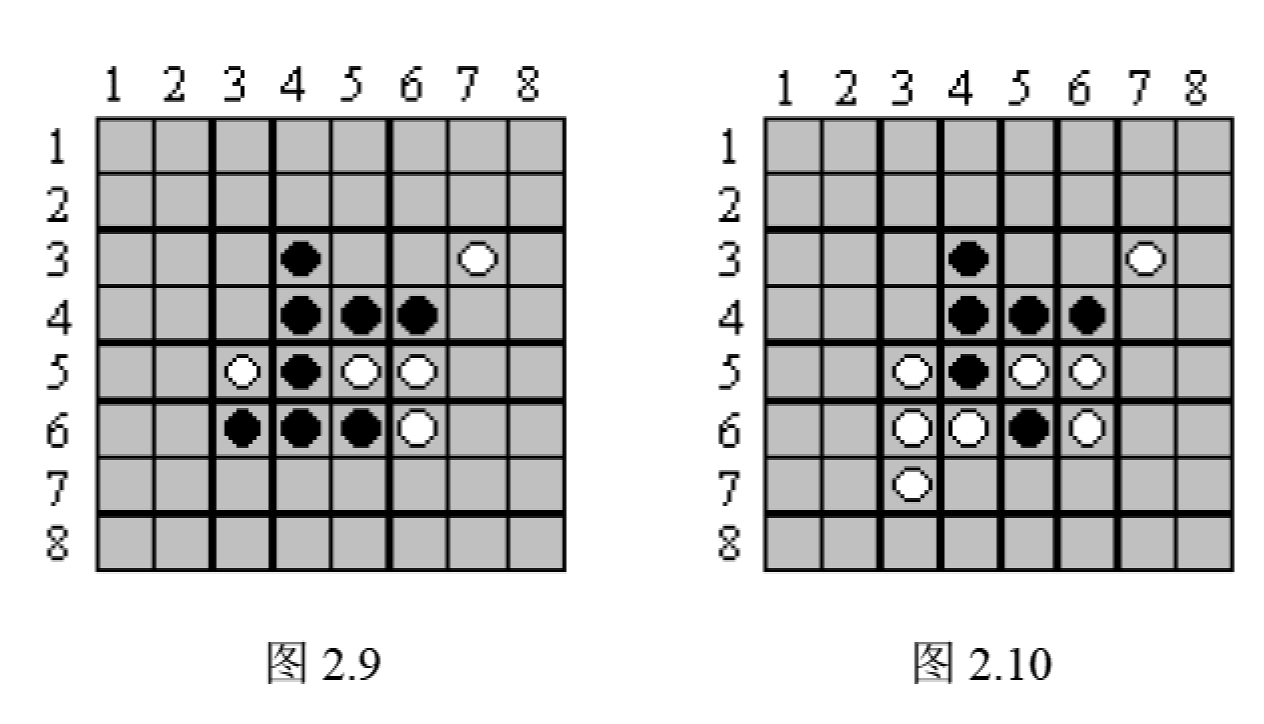
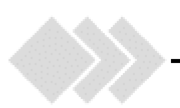


图 2.9　　图 2.10

输入一个 8×8 棋盘以及当前下一次操作的游戏者，处理以下 3 种指令：

- L 指令打印所有合法操作，按照从上到下、从左到右的顺序排列（没有合法操作时输出 No legal move）。
- Mrc 指令放一枚棋子在(r,c)。如果当前游戏者没有合法操作，则是先切换游戏者再操作。输入保证这个操作是合法的。输出操作完毕后黑白双方的棋子总数。
- Q 指令退出游戏，并打印当前棋盘（格式同输入）。



【分析】

直接进行模拟即可，需要注意的是，可以将棋子移动的方向预先定义成 8 个向量，然后使用之前提到过的坐标和向量的运算。

习题 4-4 骰子涂色（Cube painting, UVa253）

输入两个骰子，判断二者是否等价。每个骰子用 6 个字母表示，如图 2.11 所示。

例如，rbgggr 和 rggbgr 分别表示如图 2.12 和图 2.13 所示的两个骰子。二者是等价的，因为如图 2.12 所示的骰子沿着竖直轴旋转 90°之后就可以得到如图 2.13 所示的骰子。

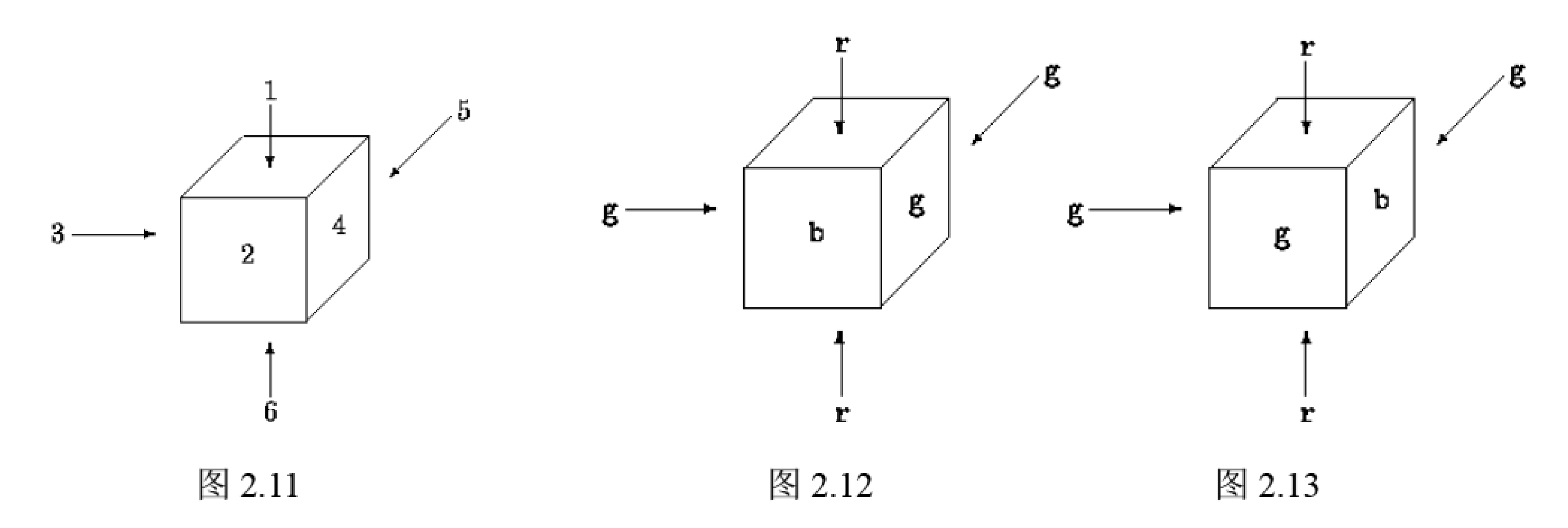


图 2.11　　图 2.12　　图 2.13

【分析】

本题可以参考《算法竞赛入门经典——训练指南》中第 1 章例题 8 的思路，通过确定顶面和正面的编号来确定所有面的编号，使用程序生成所有面的 24 种排列。枚举输入的第一个立方体的所有姿态的编码，逐一和第二个骰子的编码比较即可。

习题 4-5 IP 网络（IP Networks, ACM/ICPC NEERC 2005, UVa1590）

可以用一个网络地址和一个子网掩码描述一个子网（即连续的 IP 地址范围）。其中，子网掩码包含 32 个二进制位，前 32−n 位为 1，后 n 位为 0，网络地址的前 32−n 位任意，后 n 位为 0。所有前 32−n 位和网络地址相同的 IP 都属于此网络。

例如，网络地址为 194.85.160.176（二进制为 11000010|01010101|10100000|10110000），子网掩码为 255.255.255.248（二进制为 11111111|11111111|11111111|11111000），则该子网的 IP 地址范围是 194.85.160.176～194.85.160.183。输入一些 IP 地址，求最小的网络（即包含 IP 地址最少的网络），包含所有这些输入地址。

例如，若输入 3 个 IP 地址：194.85.160.177、194.85.160.183 和 194.85.160.178，包含上述 3 个地址的最小网络的网络地址为 194.85.160.176，子网掩码为 255.255.255.248。

【分析】

首先需要将输入的 IP 地址都转换成二进制表示，然后得到这些二进制数的最长公共前缀 P 的长度 L。则最小网络 IP 的前 L 位就是 P，剩余的位都是 0。子网掩码的前 L 位都是 1，剩余都是 0。网络地址从二进制转换到十进制的过程可以提取成公共函数，在输出结果时复用。完整程序如下：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
#define _rep(i,a,b) for( int i=(a); i<=(b); ++i)
```

```
using namespace std;

//x in [left, right]
bool inRange(int x, int left, int right) {
   if(left > right) return inRange(x, right, left);
   return left <= x && x <= right;
}

const int W = 8, IPW = 4*W;
void printIp(const int *v) {
   bool first = true;
   _for(i, 0, 4) {
      int x = 0;
      _for(j, i*W, (i+1)*W) x = (x<<1)|v[j];
      if(first) first = false; else printf(".");
      printf("%d", x);
   }
   puts("");
}

void toBinary(int x, int* v, int pos) {
   assert(inRange(x, 0, 255));
   _for(i, 0, W) v[pos+W-i-1] = x%2, x/=2;
}

const int MAXM = 1024;
int ips[MAXM][IPW + 4];

int main(){
   int m, ip[4], subNet[IPW];
   while(scanf("%d", &m) == 1) {
      memset(subNet, 0, sizeof(subNet));
      _for(i, 0, m){
         scanf("%d.%d.%d.%d", &ip[0], &ip[1], &ip[2], &ip[3]);
         _for(j, 0, 4) toBinary(ip[j], ips[i], j*W);
      }

      int n;
      for(n = 0; n < IPW; n++){
         bool same = true;
         _for(j, 1, m) if(ips[j][n] != ips[j-1][n]) { same = false; break; }
         if(!same) break;
```



```
        }

        fill_n(subNet, n, 1);
        fill_n(ips[0] + n, IPW - n, 0);
        printIp(ips[0]);
        printIp(subNet);
    }
    return 0;
}
```

习题 4-6　莫尔斯电码（Morse Mismatches, ACM/ICPC World Finals 1997, UVa508）

输入每个字母的 Morse 编码、一个词典以及若干个编码。对于每个编码，判断它可能是哪个单词。如果有多个单词精确匹配，任选一个输出并且后面加上“!”；如果无法精确匹配，可以在编码尾部增加或删除一些字符以后匹配某个单词（增加或删除的字符应尽量少）。如果有多个单词可以这样匹配上，任选一个输出并且在后面加上“?”。

莫尔斯电码的细节参见原题。

【分析】

输入时，首先建立字符到对应 Morse 编码的映射 map。每输入一个单词，通过这个 map 将每一个字符翻译成 Morse 编码，然后建立所有 Morse 编码到对应单词的映射 map<string ,vector<string> > context。

然后对于每一个输入的 Morse 编码 M，首先在 context 中查找 M 对应的所有可能的单词 v。如果 v 中只有一个单词，则输出这个单词即可；如果 v 中包含多个单词，则任意输出一个再加“!”。

如果不存在对应的 v，则查找 context 中所有符合以下条件的 Morse 编码 CM：CM 为 M 的前缀或者 M 为 CM 的前缀。找到其中长度和 M 相差最小的那个 CM 输出即可。找到的所有 CM 可以用 map<int, string>存放，key 为 CM 和 M 的大小的差，value 就是 CM 本身。因为 map 本身就是根据 key 来排序的，直接输出第一个元素即可。完整程序（C++11）如下：

```
using namespace std;

unordered_map<char, string> morse;
unordered_map<string, vector<string> > context;

//a 是 b 的前缀
bool isPrefixOf(const string& a, const string& b){
    return a.size() < b.size() && b.compare(0, a.size(), a) == 0;
}

void solve(const string& m) {
    if(context.count(m)) {
        const auto& v = context[m];
```

```
        assert(!v.empty());
        cout<<v.front();
        if(v.size() > 1) cout<<"!";
        cout<<endl;
        return;
    }

    map<int, string> ans;
    for(const auto& p : context) {
        const string& cm = p.first;
        if(isPrefixOf(m, cm)) ans[cm.size() - m.size()] = p.second.front();
        else if(isPrefixOf(cm, m)) ans[m.size() - cm.size()] = p.second.
front();
    }
    cout<<ans.begin()->second<<"?"<<endl;
}

int main(){
    string C, M;
    while(cin>>C && C != "*"){
        cin>>M;
        assert(C.size() == 1);
        morse[C[0]] = M;
    }

    while(cin>>C && C != "*") {
        M.clear();
        for(auto c : C) M += morse[c];
        context[M].push_back(C);
    }

    while(cin>>M && M != "*") solve(M);

    return 0;
}
```

习题 4-7　RAID 技术（RAID!, ACM/ICPC World Finals 1997, UVa509）

RAID 技术用多个磁盘保存数据。每份数据不止在一个磁盘上保存，因此在某个磁盘损坏时能通过其他磁盘恢复数据。本题讨论其中一种 RAID 技术。数据被划分成大小为 s（$1 \leqslant s \leqslant 64$）比特的数据块保存在 d（$2 \leqslant d \leqslant 6$）个磁盘上。如图 2.14 所示，每 d-1 个数据块都有一个校验块，使得每 d 个数据块的异或结果为全 0（偶校验）或者全 1（奇校验）。



Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
Parity for 1-4	Data block 1	Data block 2	Data block 3	Data block 4
Data block 5	**Parity for 5-8**	Data block 6	Data block 7	Data block 8
Data block 9	Data block 10	**Parity for 9-12**	Data block 11	Data block 12
Data block 13	Data block 14	Data block 15	**Parity for 13-16**	Data block 16
Data block 17	Data block 18	Data block 19	Data block 20	**Parity for 17-20**
Parity for 21-24	Data block 21	Data block 22	Data block 23	Data block 24
Data block 25	**Parity for 25-28**	Data block 26	Data block 27	Data block 28

图 2.14

例如 d=5，s=2，偶校验，数据 6C7A79EDFC（二进制 01101100 01111010 01111001 11101101 11111100）将这样保存，如图 2.15 所示。

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
00	01	10	11	00
01	**10**	11	10	10
01	11	**01**	10	01
11	10	11	**11**	01
11	11	11	00	**11**

图 2.15

其中加粗块是校验块。输入 d, s, b 校验的种类（E 表示偶校验，O 表示奇校验）以及 b（1≤b≤100）个数据块（其中“?”表示损坏的数据），你的任务是恢复并输出完整的数据。如果校验错或者由于损坏数据过多无法恢复，应报告磁盘非法。

提示：

如果没有 RAID 的知识背景，上述简要翻译可能较难理解，细节建议参考原题。

【分析】

因为输入是按照每个 Disk 依次进行，所以需要把每个 Disk 看成行，这个与题图的行列是反过来的，这一点需要注意。

本题的本质就是要对每一列的 01 数据进行还原，首先是要检查每一列是否有多个未知数据，如果多于 1 个，则无法还原，磁盘非法。对于全是已知数据的列，所有数据的异或运算结果必须符合校验种类：E 时为 0，O 时为 1。如果不符合，表示校验错误。

如果数据全部合法，则按照列的顺序对修复过的数据进行依次组合，注意校验块所在的行是依次循环递增的，在组合时要注意忽略校验块。另外，如果组合后的数据位数不是 4 的倍数，需要进行补 0 操作。

在组合的过程中，可以使用 bitset 来存储每 4 位的数据结果，bitset.to_ulong()可以方便

地把结果转换成整数来输出。注意 bitset 的第 0 位是从右边算起，而我们组合时需要从左边算起，需要进行处理。

习题 4-8　特别困的学生（Extraordinarily Tired Students, ACM/ICPC Xi'an 2006, UVa12108）

课堂上有 n 个学生（$n \leqslant 10$）。每个学生都有一个“睡眠-清醒”周期，其中第 i 个学生醒 A_i 分钟后睡 B_i 分钟，然后重复（$1 \leqslant A_i, B_i \leqslant 5$），初始时第 i 个学生处在他的周期的第 C_i 分钟。每个学生在临睡前会察看全班睡觉人数是否严格大于清醒人数，若是才睡，否则再清醒 A_i 分钟。问经过多长时间后全班都清醒。如果用（A,B,C）描述学生，图 2.16 描述了 3 个学生(2,4,1)、(1,5,2)和(1,4,3)在每个时刻的行为。

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		😴	😴	😴	😴			😴	😴	😴	😴						
😴	😴	😴	😴	😴		😴	😴	😴	😴	😴		😴	😴	😴	😴	😴	
😴	😴	😴		😴	😴	😴	😴		😴	😴	😴	😴					

图 2.16

☞**注意：**

有可能并不存在“全部都清醒”的时刻，此时应输出-1。

【分析】

可以用一个循环来模拟每分钟的行为，首先看看是否还有睡着的，如果没有，直接输出结果即可，然后依次判断每个学生的行为并且进行模拟。

问题是整体状态有可能进入死循环，为了避免这种情况，可以将每个学生的状态放在一起编码成字符串，然后使用 set 判重，如果发现重复，说明不会再清醒了，输出-1 即可。完整程序如下：

```
using namespace std;

struct Stu { int a, b, c; };
istream& operator>>(istream& is, Stu& s) { return is>>s.a>>s.b>>s.c; }

int n;
Stu stus[10+2];
set<string> es;

void encode(string& ans) {
    ans.clear();
    _for(i, 0, n) ans += (char)(stus[i].c + '0');
    return;
}
bool action(int kase, int t) {
```

```
    string e;
    encode(e);
    if(es.count(e)) {
        cout<<"Case "<<kase<<": -1"<<endl;
        return true;
    }
    es.insert(e);

    int wake = 0, sleep = 0;
    _for(i, 0, n) if(stus[i].c <= stus[i].a) wake++;
    sleep = n - wake;

    if(sleep == 0) {
        cout<<"Case "<<kase<<": "<<t<<endl;
        return true;
    }

    _for(i, 0, n){
        Stu& s = stus[i];
        s.c++;
        if(s.c == s.a + s.b + 1) s.c = 1;
        if(s.c == s.a + 1 && wake >= sleep) s.c = 1;
    }

    return false;
}

int main(){
    int k = 1;
    while(cin>>n && n) {
        es.clear();
        _for(i, 0, n) cin>>stus[i];
        int t = 1;
        while(true) if(action(k, t++)) break;
        k++;
    }
    return 0;
}
```

习题 4-9　数据挖掘（Data Mining, ACM/ICPC NEERC 2003, UVa1591）

有两个 n 元素数组 P 和 Q。P 数组每个元素占 S_P 个字节，Q 数组每个元素占 S_Q 个字节。有时需直接根据 P 数组中某个元素 P(i)的偏移量 $P_{ofs}(i)$算出对应的 Q(i)的偏移量 $Q_{ofs}(i)$。当

两个数组的元素均为连续存储时 $Q_{ofs}(i)=P_{ofs}(i)/S_p*S_q$，但因为除法慢，可以把式子改写成速度较快的 $Q_{ofs'}(i)=(P_{ofs}(i)+P_{ofs}(i)<<A)>>B$。为了让这个式子成立，在 P 数组仍然连续存储的前提下，Q 数组可以不连续存储（但不同数组元素的存储空间不能重叠）。这样做虽然会浪费一些空间，但是提升了速度，是一种用空间换时间的方法。

输入 N、S_P 和 S_Q（$N\leqslant 2^{20}$，$1\leqslant S_P, S_Q\leqslant 2^{10}$），你的任务是找到最优的 A 和 B，使得占的空间 K 尽量小。输出 K、A、B 的值。多解时让 A 尽量小，如果仍多解则让 B 尽量小。

提示：

本题有一定实际意义，不过描述比较抽象。如果对本题兴趣不大，可以先跳过。

【分析】

注意是对 32 位整数进行位移操作，那么 A 和 B 必然是 0～31 的整数。所有的 A、B 组合就只有 32*32 个，完全可以使用暴力方式遍历求解。

另外，Q 的不同元素不可以重叠，这就要求 $Q_{ofs'}(i)>i*S_q$。只需考虑 i=1 的情况就可以确定 Q 中一个元素的存储空间，也就是说要求 $Q_{ofs'}(1)=(S_p+S_p<<A)>>B\geqslant S_q$，找到一组符合要求的 A、B 之后即可得：$K = Q_{n-1}+S_q=(S_p*(n-1) + (S_p*(n-1))<<A)>>B+S_q$。这里 Q 的最后一个元素之后不需要多余的存储空间。

从小到大遍历 A 和 B，发现更小的 K 时更新答案，即可得到题目要求的解。

习题 4-10　洪水！（Flooded! ACM/ICPC World Finals 1999, UVa815）

有一个 $n*m$（$1\leqslant m, n<30$）的网格，每个格子是边长 10 米的正方形，网格四周是无限大的墙壁。输入每个格子的海拔高度，以及网格内雨水的总体积 v，输出水位的海拔高度以及有多少百分比的区域有水（即高度严格小于水平面）。

【分析】

原题保证了水会从海拔最低的格子开始淹，然后依次上升。可以把所有 $n*m$ 个格子按照海拔从低到高排列，编号从 0 开始计。从 i=1 到 $n*m$，依次计算，如果体积为 v 的水把编号 0～i−1 的每个格子都淹没所形成的水位高度 wl，如果 wl<格子 i 的高度，则说明水无法淹没格子 i。这样就得到了水位高度 wl 和有水的格子个数 i，求百分比即可。因为格子边长是 10，可以将 v 除以 100，然后把格子边长作为 1 来处理，这样每个格子底面积就为 1，方便计算。完整程序如下：

```
using namespace std;

int main(){
    vector<int> H;
    int m, n; double wl, k, v;
    for(int r = 1; scanf("%d%d", &m, &n) == 2 && m && n; r++){
        n *= m; H.clear();
        _for(i, 0, n) H.push_back(readInt());
        H.push_back(INT_MAX);
        sort(H.begin(), H.end());
```

```
        scanf("%lf", &v); v /= 100;          //抵消每个格子的底面积 10*10
        _rep(i, 1, n){ //wl: 把 0~i-1 个格子都淹没所形成的高度
            v += H[i-1], wl = v/i;
            if(wl < H[i]){ k = i; break; }   //淹没不了格子 i
        }

        printf("Region %d\nWater level is %.2lf meters.\n", r, wl);
        printf("%.2lf percent of the region is under water.\n\n",
k*100/(H.size()-1));
    }
    return 0;
}
```

2.3 C++与 STL 入门

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 5 章。

本章的习题主要是为了练习 C++语言以及 STL，程序本身并不一定很复杂。建议读者至少完成 8 道习题。如果想达到更好的效果，建议完成 12 题或以上。

习题 5-1 代码对齐（Alignment of Code, ACM/ICPC NEERC 2010, UVa1593）

输入若干行代码，要求各列单词的左边界对齐且尽量靠左。单词之间至少要空一格。每个单词不超过 80 个字符，每行不超过 180 个字符，一共最多 1000 行，如图 2.17 所示。

<table>
<tr><th>样例输入</th><th>样例输出</th></tr>
<tr><td><pre> start: integer;   // begins here
stop: integer; // ends here
 s: string;
c: char; // temp</pre></td><td><pre>start: integer; // begins here
stop: integer; // ends   here
s: string;
c: char;   // temp</pre></td></tr>
</table>

图 2.17

【分析】

可以将所有的单词看作一个表格，则每一行被空格串切分成多个列。每一列的宽度就是表格中当前列的最长单词的宽度。输出时，列与列之间用一个空格分开。

建议使用 STL 的 I/O 流，读一行使用 getline(cin, line)，其中 line 是 string。对 line 的切分，可以使用 stringstream。切分过程中，更新每一列单词的最大长度。输出时可用 setw 来设置一个输出对象的宽度。具体可以查询 STL 使用手册。完整程序如下：

```
using namespace std;

const int MAXN = 1024;
vector<string> LineWords[MAXN];
```

```
size_t WordLen[MAXN], MaxWords, LineCnt;

int main(){
    string line, word;
    MaxWords = 0; LineCnt = 0;
    fill_n(WordLen, MAXN, 0);
    while(getline(cin, line)) {
        stringstream ss(line);
        size_t wi = 0;
        while(ss>>word) {
            WordLen[wi] = max(WordLen[wi], word.size());
            wi++;
            LineWords[LineCnt].push_back(word);
        }
        MaxWords = max(MaxWords, wi);
        LineCnt++;
    }

    _for(i, 0, LineCnt){
        const auto& ws = LineWords[i];
        _for(j, 0, ws.size()) cout<<left<<setw(j < ws.size()-1 ? WordLen[j]+1 :
0)<<ws[j];
        cout<<endl;
    }

    return 0;
}
```

习题 5-2　Ducci 序列（Ducci Sequence, ACM/ICPC Seoul 2009, UVa1594）

对于一个 n 元组$(a_1, a_2, \cdots, a_n)$，可以对每个数求出它和下一个数的差的绝对值，得到一个新的 n 元组$(|a_1-a_2|, |a_2-a_3|, \cdots, |a_n-a_1|)$。重复这个过程，得到的序列称为 Ducci 序列，例如：

(8, 11, 2, 7) → (3, 9, 5, 1) → (6, 4, 4, 2) → (2, 0, 2, 4) → (2, 2, 2, 2) → (0, 0, 0, 0)

也有的 Ducci 序列会一直循环。输入 n 元组（$3 \leqslant n \leqslant 15$），你的任务是判断它最终会变成 0 还是会一直循环。输入保证最多 1000 步就会变成 0 或者循环。

【分析】

序列可以直接使用 vector<int>模拟，vector 本身已经重载了等号运算符对两个容器的内容进行比较。所以判重和判是否全 0 直接用“==”操作符即可，判重可以使用 set < vector<int> >。完整程序如下：

```
using namespace std;
```

```
int readint() { int x; scanf("%d", &x); return x;}

int main(){
    int T = readint();
    vector<int> seq, zeroSeq;
    set< vector<int> > seqs;
    while(T--) {
        int n = readint();
        seq.clear(), zeroSeq.resize(n);
        _for(i, 0, n) seq.push_back(readint());

        bool zero = false, loop = false;
        seqs.clear(), seqs.insert(seq);
        do {
            if(seq == zeroSeq) { puts("ZERO"); break; }

            int a0 = seq[0];
            _for(i, 0, n) {
                if(i == n-1) seq[i] = abs(seq[i] - a0);
                else seq[i] = abs(seq[i] - seq[i+1]);
            }
            if(seqs.count(seq)) { puts("LOOP"); break; }
            seqs.insert(seq);
        } while(true);
    }

    return 0;
}
```

习题 5-3 卡片游戏（Throwing cards away I, UVa10935）

桌上有 n（$n\leqslant 50$）张牌，从第一张牌（即位于顶面的牌）开始从上往下依次编号为 1～n。当至少还剩两张牌时进行以下操作：把第一张牌扔掉，然后把新的第一张放到整叠牌的最后。输入每行包含一个 n，输出每次扔掉的牌，以及最后剩下的牌。

【分析】

因为对牌堆的操作就是出队和入队，直接使用 STL 里面的 queue 来模拟牌堆即可。需要注意的特殊情况是，当 n=1 时，直接输出“Discarded cards:”，行尾是不包含空格的。完整程序如下：

```
using namespace std;

int main(){
    int n;
```

```
    while(scanf("%d", &n) == 1 && n) {
        queue<int> q;
        _rep(i, 1, n) q.push(i);
        printf("Discarded cards:");
        bool first = true;
        while(q.size() >= 2) {
            if(first) { first = false; printf(" %d", q.front()); }
            else printf(", %d", q.front());
            q.pop();
            q.push(q.front());
            q.pop();
        }
        printf("\nRemaining card: %d\n", q.front());
    }
    return 0;
}

/* 特殊情况：n=1 时，
Discarded cards:<- No space here!!!
Remaining card: 1 */
```

习题 5-4　交换学生（Foreign Exchange, UVa10763）

有 n（$1 \leqslant n \leqslant 500000$）个学生想交换到其他学校学习。为了简单起见，规定每个想从 A 学校换到 B 学校的学生必须找一个想从 B 换到 A 的“搭档”。如果每个人都能找到自己的搭档（一个人不能当多个人的搭档），学校就会同意他们交换。每个学生用两个整数 A、B 表示，你的任务是判断交换是否可以进行。

【分析】

对于每所学校，如果要交换成功，必须要出去和要进来的人数完全相等。可以在输入的同时记每一个 A 和 B 组成的整数对，以及对应的 A→B 的交换学生个数。

输入完成后，遍历所有的（A,B），看看对应的学生个数是否和（B,A）的相同，若有不同，则交换不能进行。完整程序（C++11）如下：

```
using namespace std;
int readint() { int x; scanf("%d", &x); return x;}
typedef pair<int, int> IPair;

int main() {
    map<IPair, int> ex; int n;
    while(n = readint()){
        bool ans = true;
        ex.clear();
        _for(i, 0, n){
```

```
            int A = readint(), B = readint();
            ex[make_pair(A,B)]++;
        }
        for(const auto& p : ex) {
            IPair p2 = make_pair(p.first.second, p.first.first);
            if(p.second != ex[p2]) { ans = false; break; }
        }
        puts(ans ? "YES" : "NO");
    }
    return 0;
}
```

习题 5-5　复合词（Compound Words, UVa 10391）

给一个词典，找出所有的复合词，即恰好有两个单词连接而成的单词。输入每行都是一个由小写字母组成的单词。输入已按照字典序从小到大排序，且不超过 120 000 个单词。输出所有复合词，按照字典序从小到大排列。

【分析】

使用 set<string>来存储所有单词，然后对每个单词的所有左右切分方案进行遍历，判断切分出来的两个子字符串两边是否同时存在于 set 中，符合条件的直接输出即可。这里 set 中的字符串已经按照默认字典序排序。完整程序如下：

```
using namespace std;

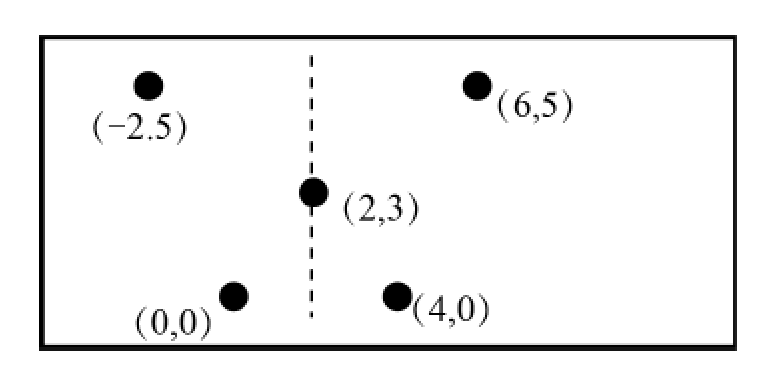
int main(){
    ios::sync_with_stdio(false);
    set<string> words;
    string word, left, right;
    while(cin>>word) words.insert(word);

    for(const auto& s : words) _for(j, 1, s.size()) {
        left.assign(s, 0, j);
        if(words.count(left)){
            right.assign(s, j, s.size() - j);
            if(words.count(right)) { cout<<s<<endl; break; }
        }
    }

    return 0;
}
```

习题 5-6　对称轴（Symmetry, ACM/ICPC Seoul 2004, UVa1595）

给出平面上 N（$N\leqslant1000$）个点，问是否可以找到一条竖线使得所有点左右对称。如图 2.18 所示，左边的图形有对称轴，右边没有。



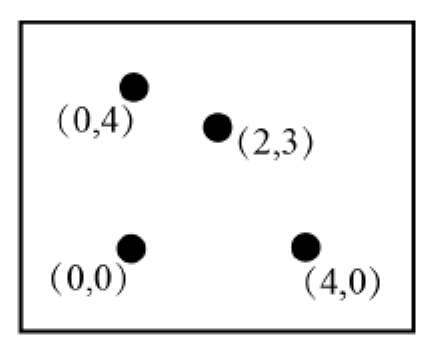


图 2.18

【分析】

如果存在对称轴 $x = m$，那么 m 一定是所有点 x 坐标的平均值。首先在输入每个点坐标的同时就求出 m。然后遍历每一个点 p，要么 $p.x = m$ 就是说 p 关于 x=m 的对称点就是自身，或者 p 关于 x=m 的对称点也在输入的点中，否则说明对称轴不存在。

在输入时就需要对所有的点建立索引，可以使用 Point 类，同时使用 set<Point>来存储所有的点，为此 Point 类需要重载“<”比较运算符。

习题 5-7　打印队列（Printer Queue, ACM/ICPC NWERC 2006, UVa12100）

学生会里只有一台打印机，但是有很多东西需要打印，因此打印任务不可避免的需要等待。有些打印任务比较急，有些不那么急，所以每个任务都有一个 1～9 的优先级，越大表示任务越急。

打印机的运作方式如下：首先从打印队列里取出一个任务 J，如果队列里有比 J 更急的任务，则直接把 J 放到打印队列尾部，否则打印任务 J（此时不会把它放回打印队列）。

输入打印队列中各个任务的优先级以及你所关注的任务在队列中的位置(队首位置为0)，输出该任务完成的时刻。所有任务都需要 1 分钟打印。例如，打印队列为{1, 1, 9, 1, 1, 1}，目前处于队首的任务最终完成时刻为 5。

【分析】

使用 STL 中的 queue 来模拟打印队列，同时因为需要判断是否有更急的任务，那么就需要使用一个 pCnt 数组来维护每个优先级 p 在当前队列中对应的任务的数量，需要在队列变化时更新这个 pCnt 数组。完整程序如下：

```
using namespace std;
int readint() { int x; scanf("%d", &x); return x; }
const int MAXN = 128, MAXP = 10;
int n, m, priority[MAXN], pCnt[MAXP];
int main()
{
    int T = readint();
    while(T--) {
        n = readint(), m = readint();
        fill_n(pCnt, MAXP, 0);
```

```
        fill_n(priority, MAXN, 0);
        queue<int> q;
        for(int i = 0; i < n; i++) {
            q.push(i);
            priority[i] = readint();
            pCnt[priority[i]]++;
        }

        int timer = 1;
        while(!q.empty()) {
            int t = q.front(), p = priority[t];
            bool lower = false;
            for(int hp = MAXP-1; hp > p; hp--)
                if(pCnt[hp] > 0) { lower = true; break; }

            q.pop();
            if(lower) { q.push(t); continue; }
            if(t == m) break;

            pCnt[p]--;
            assert(pCnt[p] >= 0);
            timer++;
        }

        printf("%d\n", timer);
    }

    return 0;
}
```

习题 5-8　图书管理系统（Borrowers, ACM/ICPC World Finals 1994, UVa230）

你的任务是模拟一个图书管理系统。首先输入若干图书的标题和作者（标题各不相同，以 END 结束），然后是若干指令：BORROW 指令借书，RETURN 指令还书。对于 SHELVE 指令，把所有已归还但还未上架的图书排序后依次插入到架子上并输出图书标题和插入位置（可能是第一本书或者某本书的后面）。

图书排序的方法是先按作者从小到大排，再按标题从小到大排。在处理第一条指令之前，你应当先将所有图书按照这种方式排序。

【分析】

首先建立一个 Book 类，使用一个 vector<Book>来保存所有的图书。同时使用 map<string, int>建立一个标题到图书编号（其在 vector 中的位置）的映射，方便根据标题来查找图书。

使用两个 set<int>来分别保存已归还但是还未上架的书籍 shelf，以及还在书架上的书籍

lib。set 的第二个泛型参数（用于指定 set 内部对于 Book 类的排序规则）要使用一个函数对象（functor）来对其中的书籍位置按照题目要求的规则进行排序。

这样借书还书时直接对 shelf 和 lib 进行操作即可。上架时，每次在 lib 中插入一个元素之后，可以取到这个元素在 set 中的位置 iterator，然后对这个 iterator 进行自减操作即可拿到插入的书籍所在位置前面的一本书。完整代码如下：

```
using namespace std;

struct Book {
   string title, author;
   Book(const string& t, const string& a) : title(t), author(a) {}
   bool operator<(const Book rhs) const { return author<rhs.author ||
(author==rhs.author && title<rhs.title); }
};

vector<Book> books;
map<string, int> bookIndice;
struct indexComp {
   bool operator() (const int& lhs, const int& rhs) const {
      return books[lhs] < books[rhs];
   }
};

set<int, indexComp> shelf, lib;

void borrow(const string& t) {
   assert(bookIndice.count(t));
   int idx = bookIndice[t];
   if(lib.count(idx)) {
      lib.erase(idx);
   }else {
      assert(shelf.count(idx));
      shelf.erase(idx);
   }
}

void retBook(const string& t) {
   assert(bookIndice.count(t));
   int idx = bookIndice[t];
   assert(!lib.count(idx));
   assert(!shelf.count(idx));
```

```
    shelf.insert(idx);
}

void shelve() {
    for(set<int>::iterator it = shelf.begin(); it != shelf.end(); it++) {
        int idx = *it;
        set<int>::iterator pit = lib.insert(idx).first;
        if(pit == lib.begin())
            cout<<"Put "<<books[idx].title<<" first"<<endl;
        else {
            pit--;
           cout<<"Put "<<books[idx].title<<" after "<<books[*pit].title<<endl;
        }
    }

    shelf.clear();
    cout<<"END"<<endl;
}

int main()
{
    string buf;
    while(true) {
        getline(cin, buf);
        if(buf == "END") break;
        int spos = buf.find(" by ");
        assert(spos != string::npos);
        string title = buf.substr(0, spos), author = buf.substr(spos + 4);
        int idx = books.size();
        //cout<<"title = "<<title<<endl;
        bookIndice[title] = idx;
        books.push_back(Book(title, author));
    }
    for(int i = 0; i < books.size(); i++)
        lib.insert(i);

    string cmd, title;
    while(true) {
        getline(cin, buf);
        if(buf == "END") break;
        cmd = buf.substr(0, 6);
        if(cmd[0] == 'S') shelve();
```

```
        else {
            title = buf.substr(cmd.size() + 1, buf.size() - cmd.size() - 1);
            if(cmd[0] == 'B') borrow(title);
            else { assert(cmd[0] = 'R'); retBook(title); }
        }
    }

    return 0;
}
```

习题 5-9　找 bug（Bug Hunt, ACM/ICPC Tokyo 2007, UVa1596）

输入并模拟执行一段程序，输出第一个 bug 所在行。每行程序有两种可能。

- 数组定义：格式为 arr[size]。如 a[10]或者 b[5]，可用下标分别是 0～9 和 0～4。定义之后所有元素均为未初始化状态。
- 赋值语句：格式为 arr[index]=value。如 a[0]=3 或者 a[a[0]]=a[1]。

赋值语句可能会出现两种 bug：下标 index 越界；使用未初始化的变量（index 和 value 都可能出现这种情况）。

程序不超过 1000 行，每行不超过 80 个字符且所有常数均为小于 2^{31} 的非负整数。

【分析】

建立一个 Array 类，里面包含数组的大小，并且用一个 map<int, int>来表示初始化过的数组下标和对应的值。用 size=−1 表示数组不存在，并且初始 map 是空的，表示每个位置上的元素均未初始化。之后在对每个语句求值时，递归求出每个表达式的值，然后再进行相应的赋值或者取值。另外，需要特别注意 array 的 size=0 也是合法的。

```
using namespace std;

struct Array {
    int size;
    map<int, int> values;
    void init(int sz) {

        assert(sz>=0);
        //printf("init size = %d\n", sz);
        size = sz;
        values.clear();
    }

    Array(){ remove(); }

    void remove() { size = -1; values.clear(); }
```

```
    bool exists() { return size >= 0; }

    bool getValue(int idx, int& v) {
        assert(exists());
        if(values.count(idx)) {
            v = values[idx];
            return true;
        }
        return false;
    }

    bool setValue(int idx, int v) {
        assert(exists());
        assert(idx >= 0);
        if(idx >= size) return false;

        values[idx] = v;
        return true;
    }
};

const int MAXA = 128;
Array arrays[MAXA];

bool eval(const char* s, int len, int& v) {
    //printf("eval %s, len = %d,\n", s, len);
    if(isdigit(s[0])) {
        sscanf(s, "%d", &v); return true;
    }

    char a = s[0];
    assert(len > 3);
    assert(isalpha(a));
    assert(s[1] == '[');
    assert(s[len-1] == ']');

    Array& ary = arrays[a];
    if(!ary.exists()) return false;
    int idx;
    if(!eval(s+2, len-3, idx)) return false;
    return ary.getValue(idx, v);
}
```

```
int main()
{
    char line[128];
    int lineNum = 0, bugLine = 0;
    while(scanf("%s", line) == 1){
        //printf("e : %s\n", line);
        int expLen = strlen(line);
        if(line[0] == '.') {
            if(lineNum) printf("%d\n", bugLine);
            for(int i = 0; i < MAXA; i++) arrays[i].remove();
            lineNum = 0;
            bugLine = 0;
            continue;
        }
        if(bugLine > 0) continue;

        const char *pEq = strchr(line, '=');
        if(pEq)
        {
            Array& ary = arrays[line[0]];
            int rv, index, lLen = pEq - line;
            if(ary.exists()
                && eval(pEq+1, expLen-lLen-1, rv)
                && eval(line+2, lLen-3, index)
                && ary.setValue(index, rv))
                lineNum++;
            else
                bugLine = lineNum+1;
        } else {
            char name; int sz;
            sscanf(line, "%c[%d]", &name, &sz);
            arrays[name].init(sz);
            lineNum++;
        }
    }

    return 0;
}
```

习题 5-10　在 Web 中搜索（Searching the Web, ACM/ICPC Beijing 2004, UVa1597）

输入 n 篇文章和 m 个请求（n<100，m≤50000），每个请求都是 4 种格式之一。



- A：找包含关键字 A 的文章。
- A AND B：找同时包含关键字 A 和 B 的文章。
- A OR B：找包含关键字 A 或 B 的文章。
- NOT A：找不包含关键字 A 的文章。

处理询问时需要对每篇文章输出证据。前 3 种询问输出所有至少包含一个关键字的行，第 4 种询问输出不包含 A 的整篇文章。关键字只由小写字母组成，查找时忽略大小写。每行不超过 80 字符，一共不超过 1500 行。

【分析】

首先在输入时需要对所有的行进行记录，把所有的行字符串放到一个 vector<string>中，后续对行的处理都通过一个整数索引来进行。

经过仔细观察可以发现，对于一个文章来说，所有的查询都可以归结为如下的操作：对于一个单词 w，查询所有包含 w 的行的序号。由此可以对每一篇文章建立一个结构 Doc，里面包含文章中所有的行的序号 vector<int> lines，以及一个 map<string, set<int> > words 保存每个单词的所在行号集合。这样在每次插入行时就可以维护 words 以备后续查询。

这个结构建立起来之后，对于 AND 和 OR 查询，就是对查询两个单词输出的两个 set<int>进行 set_union 即可。对于 NOT 查询，符合条件的文章就是查询出来的结果为空的那些。对于每个单词查询，符合条件的文章就是查询出集合不为空的那些。

另外一个技巧就是，针对 set<int>提供一个 operator<<的运算符重载，方便对查询出来的行进行输出。具体请参照以下代码：

```
using namespace std;

typedef vector<int> IntVec;
typedef set<int> IntSet;

IntSet emptyIntSet;

struct Doc {
    IntSet lines;
    map<string, IntSet> words;

    void AddLine(const string& s, int l) {
        lines.insert(l);
        string w;
        for(int i = 0; i < s.size(); i++) {
            char c = s[i];
            if(isalpha(c)) w.push_back(tolower(c));
            else if(!w.empty()) { words[w].insert(l); w.clear(); }
        }
        if(!w.empty()) { words[w].insert(l); }
```

```
        }

        const IntSet& FindWord(const string& w) {
            if(!words.count(w)) return emptyIntSet;
            return words[w];
        }
    };

    vector<Doc> docs;
    vector<string> Lines;

    ostream& operator<<(ostream& os, const IntSet& lines) {
        for(IntSet::const_iterator cit = lines.begin(); cit != lines.end();
cit++)
            os<<Lines[*cit]<<endl;
        return os;
    }

    void parseQuery(const string& q, vector<string>& ws) {
        ws.clear();
        stringstream ss(q);
        string w;
        while(ss>>w) ws.push_back(w);
    }

    void doQuery(const vector<string>& qWs) {
        assert(!qWs.empty());
        const string& A = qWs.front();
        const string& B = qWs.back();
        bool isAnd = (qWs.size()==3 && qWs[1]=="AND"), first = true, match;
        stringstream ss;
        switch(qWs.size()) {
            case 1:
                for(int i = 0; i < docs.size(); i++) {
                    Doc& a = docs[i];
                    const IntSet& ans = a.FindWord(A);
                    match = !ans.empty();
                    if(!match) continue;

                    if(first) first = false; else ss<<"----------"<<endl;
                    ss<<ans;
                }
```

```
            break;
        case 2:
            assert(A == "NOT");
            for(int i = 0; i < docs.size(); i++) {
                Doc& a = docs[i];
                const IntSet& ans = a.FindWord(B);
                match = ans.empty();
                if(!match) continue;

                if(first) first = false; else ss<<"----------"<<endl;
                ss<<a.lines;
            }
            break;
        case 3:
            assert(isAnd || (qWs[1] == "OR"));
            for(int i = 0; i < docs.size(); i++) {
                Doc& a = docs[i];
                const IntSet& ansA = a.FindWord(A);
                const IntSet& ansB = a.FindWord(B);
                if(isAnd) match = (!ansA.empty()) && (!ansB.empty());
                else match = (!ansA.empty()) || (!ansB.empty());
                if(!match) continue;

                IntVec ans(ansA.size() + ansB.size());
                IntVec::iterator st = set_union(ansA.begin(), ansA.end(),
                    ansB.begin(), ansB.end(), ans.begin());

                if(first) first = false; else ss<<"----------"<<endl;
                for(IntVec::iterator it = ans.begin(); it != st; it++)
                    ss<<Lines[*it]<<endl;
            }
            break;
        default:
            assert(false);
            break;
    }
    const string& output = ss.str();
    if(output.empty()) cout<<"Sorry, I found nothing."<<endl;
    cout<<output<<"=========="<<endl;
}

int main()
```

```
{
    ios::sync_with_stdio(false);
    int N, M;
    string line;
    cin>>N;
    getline(cin, line);
    docs.resize(N);

    for(int i = 0; i < N; i++) {
        Doc& d = docs[i];
        while(true) {
            getline(cin, line);
            if(line == "**********") break;
            Lines.push_back(line);
            d.AddLine(line, Lines.size()-1);
        }
    }

    cin>>M;
    getline(cin, line);
    vector<string> qWs;
    for(int i = 0; i < M; i++) {
        getline(cin, line);
        parseQuery(line, qWs);
        doQuery(qWs);
    }

    return 0;
}
```

习题 5-11　更新字典（Updating a Dictionary, UVa12504）

在本题中，字典是若干键值对，其中键为小写字母组成的字符串，值为没有前导零或正号的非负整数（因此-4、03 和+77 都是非法的。注意该整数可以很大）。输入一个旧字典和一个新字典，计算二者的变化。输入的两个字典中键都是唯一的，但是排列顺序任意。具体格式如下（注意字典格式中不含任何空白字符）：

```
{key:value,key:value,…,key:value}
```

输入包含两行，各包含不超过 100 个字符，即旧字典和新字典。输出格式如下。

- 如果至少有一个新增键，打印一个“+”号，然后是所有新增键，按字典序从小到大排列。
- 如果至少有一个删除键，打印一个“-”号，然后是所有删除键，按字典序从小到大排列。

- ❑ 如果至少有一个修改键，打印一个“*”号，然后是所有修改键，按字典序从小到大排列。
- ❑ 如果没有任何修改，输出 No changes。

例如，若输入两行分别为{a:3,b:4,c:10,f:6}和{a:3,c:5,d:10,ee:4}，输出为以下 3 行：+d,ee；−b,f；*c。

【分析】

对于每一行，使用一个线性遍历把数据解析成 map<string, string>，然后对两个 map 进行对比，就可以判断出所求的 3 种键。还建议对 vector<string>重载 operator<<操作符，方便对比较的结果进行输出：

```
ostream& operator<<(ostream& os, const vector<string>& s) {
   bool first = true;
   for(int i = 0; i < s.size(); i++) {
      if(first) first = false;
      else os<<",";
      os<<s[i];
   }
   return os;
}
```

输出结果的代码就是这样的：

```
if(!added.empty()) cout<<"+"<<added<<endl;
if(!deled.empty()) cout<<"-"<<deled<<endl;
if(!changed.empty()) cout<<"*"<<changed<<endl;
```

习题 5-12　地图查询（Do You Know The Way to San Jose?, ACM/ICPC World Finals 1997, UVa511）

有 n 张地图（已知名称和某两个对角线端点的坐标）和 m 个地名（已知名称和坐标），还有 q 个查询。每张地图都是边平行于坐标轴的矩形，比例定义为高度除以宽度的值。每个查询包含一个地名和详细等级 i。假定包含此地名的地图中一共有 k 种不同的面积，则合法的详细等级为 1～k（其中 1 最不详细，k 最详细，面积越小越详细）。如果详细等级 i 的地图不止一张，则输出地图中心和查询地名最接近的一张；如果还有并列的，地图长宽比应尽量接近 0.75（这是 Web 浏览器的比例）；如果还有并列，查询地名和地图右下角坐标应最远（对应最少的滚动条移动）；如果还有并列，则输出 x 坐标最小的一个。如果查询的地名不存在或者没有地图包含它，或者包含它的地图总数超过 i，应报告查询非法（并且输出包含它的最详细地图名称，如果有的话）。

提示：

本题的要求比较细致，如果打算编程实现，建议参考原题。

【分析】

因为牵涉比较多的二维几何计算，可以使用 Point 类来简化相关代码。首先是定义 Map 类，包含所有的相关信息（ratio, width, height, area, minx 最小 x 坐标）等，并且需要对所有的地名的坐标建立索引，使用 map<string, Point>。

对于每个具体的查询，首先是根据坐标查出包含这个坐标的所有 map 以及相应的 level 值，之后需要对这些 map 进行排序，排序规则如题意所示。首先按照 level（也就是面积从小到大）进行排序，level 相同按照其他规则进行排序。排序规则可以封装到一个 STL 的 functor 对象中去。完整程序如下：

```
using namespace std;

struct Point {
   double x, y;
   Point(double x=0, double y=0):x(x),y(y) {}
};

typedef Point Vector;
const double eps = 1e-7;
int dcmp(double x) { if(fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
int cmp(double x, double y) { return dcmp(x-y); }

//x in [left, right]
bool inRange(double x, double l, double r) {
   if(cmp(l, r) > 0) return inRange(x, r, l);
   return cmp(l, x) <= 0 && cmp(x, r) <= 0;
}

bool inArea(const Point& p, const Point& l, const Point& r) {
   return inRange(p.x, l.x, r.x) && inRange(p.y, l.y, r.y);
}

Vector operator + (const Vector& A, const Vector& B) { return Vector(A.x+B.x,
A.y+B.y); }
Vector operator - (const Point& A, const Point& B) { return Vector(A.x-B.x,
A.y-B.y); }
Vector operator * (const Vector& A, double p) { return Vector(A.x*p, A.y*p); }
double Dot(const Vector& A, const Vector& B) { return A.x*B.x + A.y*B.y; }
double Dist2(const Point& A, const Point& B) { return Dot(A-B,A-B); }
double Length(const Vector& A) { return sqrt(Dot(A, A)); }

istream& operator>> (istream& is, Point& p) { return is>>p.x>>p.y; }

struct Map {
   string name;
```

```
    Point corner1, corner2, center, lowerRight;
    double ratio, width, height, area, minX;
    void init() {
        center = (corner1 + corner2) * .5;
        width = fabs(corner1.x - corner2.x);
        height = fabs(corner1.y - corner2.y);
        ratio = fabs(height/width - 0.75);
        area = width * height;
        lowerRight.x = center.x + width/2;
        lowerRight.y = center.y - height/2;
        minX = center.x - width/2;
    }
};

vector<Map> maps;
map<string, Point> locIndice;

struct mapComp {
    Point loc;

    bool operator() (int i1, int i2) {
        const Map& m1 = maps[i1];
        const Map& m2 = maps[i2];
        int cr;
        //area compare
        cr = cmp(m1.area, m2.area);
        if(cr > 0) return true;
        if(cr < 0) return false;        //面积小的往后排

        //location is nearest the center of the map.
        cr = cmp(Dist2(loc, m1.center), Dist2(loc, m2.center));
        if(cr > 0) return true;
        if(cr < 0) return false;

        //aspect ratio is nearest to the aspect ratio of the browser window,
which is 0.75.
        cr = cmp(m1.ratio, m2.ratio);
        if(cr > 0) return true;
        if(cr < 0) return false;

        //which the location is furthest from the lower right corner of the map
        cr = cmp(Dist2(loc, m1.lowerRight), Dist2(loc, m2.lowerRight));
        if(cr < 0) return true;
        if(cr > 0) return false;
```

```
            /*
            one containing the smallest x-coordinate.
            */
            cr = cmp(m1.minX, m2.minX);
            assert(!cr);

            if(cr > 0) return true;
            if(cr < 0) return false;

            return true;
        }
    };

    void getMaps(const Point& p, vector<int>& mis, vector<int>& level) {
        mis.clear();
        mapComp mc;
        mc.loc = p;
        for(int i = 0; i < maps.size(); i++) {
            const Map& m = maps[i];
            if(inArea(p, m.corner1, m.corner2)) mis.push_back(i);
        }
        sort(mis.begin(), mis.end(), mc);
        level.clear();
        level.assign(mis.size(), 1);

        //cout<<endl;
        for(int i = 0; i < mis.size(); i++) {
            if(!i) continue;
            const Map& m = maps[mis[i]];
            const Map& pm = maps[mis[i-1]];
            int c = cmp(m.area, pm.area);
            assert(c <= 0);
            level[i] = level[i-1];
            if(c<0) level[i]++;
            //cout<<"name: "<<m.name<<", area : "<<m.area<<", level : "<<level[i]
<<endl;
        }
    }

    void doRequest(const string& name, int level) {
        cout<<name<<" at detail level "<<level;
```

```
    if(!locIndice.count(name)) {
        cout<<" unknown location"<<endl;
        return;
    }

    vector<int> mis, levels;
    getMaps(locIndice[name], mis, levels);
    if(mis.empty()) {
        cout<<" no map contains that location"<<endl;
        return;
    }

    int maxLevel = levels.back();
    if(maxLevel < level)
         cout<<" no map at that detail level; using "<<maps[mis.back()].
name<<endl;
    else
    {
         vector<int>::iterator it = upper_bound(levels.begin(), levels.end(),
level);
         assert(it != levels.begin());
         cout<<" using "<<maps[mis[it-levels.begin()-1]].name<<endl;
    }
}

int main(){
    string buf;
    getline(cin, buf);
    while(true) {
        Map m;
        cin>>m.name;
        if(m.name == "LOCATIONS") break;
        cin>>m.corner1>>m.corner2;
        m.init();
        maps.push_back(m);
    }
    Point loc;
    string name;
    while(true) {
        cin>>name;
        if(name == "REQUESTS") break;
        cin>>loc;
```

```
            locIndice[name] = loc;
        }

        while(true) {
            cin>>name;
            if(name == "END") break;
            int level;
            cin>>level;
            doRequest(name, level);
        }

        return 0;
    }
```

习题 5-13　客户中心模拟（Queue and A, ACM/ICPC World Finals 2000, UVa822）

你的任务是模拟一个客户中心运作情况。客服请求一共有 n（$1 \leqslant n \leqslant 20$）种主题，每种主题用 5 个整数描述：tid、num、t0、t 和 dt，其中 tid 为主题的唯一标识符，num 为该主题的请求个数，t0 为第一个请求的时刻，t 为处理一个请求的时间，dt 为相邻两个请求之间的间隔（为了简单情况，假定同一个主题的请求按照相同的间隔到达）。

客户中心有 m（$1 \leqslant m \leqslant 5$）个客服，每个客服用至少 3 个整数描述：tid, k, tid_1, tid_2, …, tid_k，表示一个标识符为 pid 的人可以处理 k 种主题的请求，按照优先级从大到小依次为 tid_1, tid_2, …, tid_k。当一个人有空时，他会按照优先级顺序找到第一个可以处理的请求。如果有多个人同时选中了某个请求，上次开始处理请求的时间早的人优先；如果有并列，id 小的优先。输出最后一个请求处理完毕的时刻。

【分析】

核心部分的逻辑是将所有要发生的事情用事件来表示，用优先级队列来维护所有的事件，循环着每次从中取出最早的一个事件，然后按照事件类型进行分类处理：

```
struct Event {
    int time, id;
    bool isRorC;            //是请求到达还是客服空闲事件
}
```

输入时，每种请求就实现生成 num 个事件放到事件队列中。模拟的循环中，每个时间点，用 multiset<int>作为要服务的请求队列，使用 multiset 是因为队列中可能有相同主题的请求。同时用一个 set 维护空闲的客服编号。

首先取出所有时间相同的队首事件，挨个进行处理。

（1）如果是请求事件，就放到请求队列。

（2）如果是客服事件，就将客服加到空闲客服集合中。

然后就是针对当前空闲客服以及请求队列中的请求进行匹配处理。

while(请求队列非空 && 空闲客服集合非空) {

（1）针对每个请求建立一个集合 set<int>，放置所有可以服务此请求的客服编号，编号排序规则参考题目的表述。

（2）先将每个客服按照优先级分配到其能处理的每个任务的集合中。

（3）如果没有进行分配，直接退出 while 循环。

（4）按照之前分配好的任务集合给客服分配任务，对于每个分配好的客服，要构造一个其变为空闲的事件，放入事件队列。

}

完整程序如下：

```
using namespace std;
const int maxn = 20 + 1, maxm = 8;

struct Event {
   int time, id;
   bool isRorC;                 //is request event or staff event
   bool operator<(const Event& e) const { return time > e.time; }
   Event(int t, int i, bool isr = true) : time(t), id(i), isRorC(isr) {
       assert(t >= 0);
    }
};
struct ReqInfo { int tid, num, t0, t, dt; };
struct StaffInfo {
   int pid, k, tids[maxn], idx, last, req;
   bool operator<(const StaffInfo& s) const {
      return last < s.last || (last == s.last && pid < s.pid);
   }
};

ReqInfo reqs[maxn];
StaffInfo staffs[maxm];
int n, m, kase;
multiset<int> rQs;              //当前队列中的所有任务
priority_queue<Event> em;       //按照时间排序的事件
set<int> freeStaffs;

struct StaffComp {
   bool operator() (int lhs, int rhs) const { return staffs[lhs] <staffs
[rhs]; }
};
set<int, StaffComp> rt[maxn];
```

```
void solve() {
    int time = em.top().time;                          //最新事件的时间
    while(!em.empty() && time == em.top().time) {      //收集所有的事件
        const Event& e = em.top();
        if(e.isRorC) rQs.insert(e.id);                 //往请求队列里面
        else freeStaffs.insert(e.id);                  //客服开始空闲了
        em.pop();
    }

    //人选请求如何选
    while(!rQs.empty() && !freeStaffs.empty()) {       //有请求并且有人
        _for(i, 0, n) rt[i].clear();
        bool canAssign = false;

        for(auto& i : freeStaffs){
            auto& si = staffs[i];
            _for(j, 0, si.k){
                int tid = si.tids[j];
                if(!rQs.count(tid)) continue;
                canAssign = true;
                rt[tid].insert(si.idx);
                break;
            }
        }

        if(!canAssign) break;

        _for(i, 0, n){
            auto& ss = rt[i];
            while(rQs.count(i) && !ss.empty()) {
                rQs.erase(rQs.find(i));
                int si = *(ss.begin());
                auto& s = staffs[si];
                s.last = time;
                em.push(Event(time + reqs[i].t, s.idx, false));
                freeStaffs.erase(s.idx);
                ss.erase(si);
            }
        }
    }

    if(em.empty())
```

```
        cout<<"Scenario "<<kase++<<": All requests are serviced within
"<<time<<" minutes."<<endl;
    }

    int main(){
        map<int, int> tids;
        kase = 1;
        while(cin>>n && n) {
            freeStaffs.clear(), tids.clear(), rQs.clear();
            _for(i, 0, n){
                auto& r = reqs[i];
                cin>>r.tid>>r.num>>r.t0>>r.t>>r.dt;
                tids[r.tid] = i;
                r.tid = i;
                _for(j, 0, r.num) em.push(Event(r.t0 + r.dt*j, i));
            }

            cin>>m;
            _for(i, 0, m){
                auto& s = staffs[i];
                cin>>s.pid>>s.k;
                _for(j, 0, s.k){
                    cin>>s.tids[j];
                    s.tids[j] = tids[s.tids[j]];
                }
                s.last = 0;
                s.idx = i;
                em.push(Event(0, s.idx, false));
            }

            while(!em.empty()) solve();
        }
        return 0;
    }
```

此类离散事件模拟类的题目最关键是要掌握事件概念的抽象方法以及优先级队列的使用。习题 5-16 也可以作为很好的练习题目。

习题 5-14　交易所（Exchange, ACM/ICPC NEERC 2006, UVa1598）

你的任务是为交易所设计一个订单处理系统。要求支持如下 3 种指令。

- BUY p q：有人想买，数量为 p，价格为 q。
- SELL p q：有人想卖，数量为 p，价格为 q。

- ❑ CANCEL i：取消第 i 条指令对应的订单（输入保证该指令是 BUY 或者 SELL）。

交易规则如下：对于当前买订单，若当前最低卖价（ask price）低于当前出价，则发生交易；对于当前卖订单，若当前最高买价（bid price）高于当前价格，则发生交易。发生交易时，按供需物品个数的最小值交易。交易后，需修改订单的供需物品个数。当出价或价格相同时，按订单产生的先后顺序发生交易。输入输出细节请参考原题。

提示：

本题是一个不错的优先队列练习题。

【分析】

表面上来看，买卖都是一个优先级队列，但是这里有一个需求就是需要随时从队列中删除一个元素，如果用 heap 实现（STL 中的 priority_queue 就是基于 heap 实现），删除的时间会比较长：需要从 vector 中删除然后重新构造 heap。

所以可使用 set<int>来实现队列，队列中保存的是 Order 的编号，先按照价格排序，再按照订单的先后顺序也就是编号的大小排序。因为两个队列的排序规则不同，将两个规则分别封装成为两个 functor 对象，然后将其作为泛型参数来声明两个不同的 set 即可。完整程序如下：

```
using namespace std;

struct Order {
   bool buy;
   int size, price;
};

ostream& operator<<(ostream& os, const Order& o)
{ return os<<"("<<o.price<<","<<o.size<<")"; }

const int MAXN = 10000 + 10;

int n, orderIndice[MAXN], canceled[MAXN];
vector<Order> orders;

template<typename Compare>
struct OrderQueue {
   typedef set<int, Compare> _IntSet;
   _IntSet eles;
   void erase(int x) { eles.erase(eles.find(x)); }
   bool empty() const { return eles.empty(); }
   int top() const { return *eles.begin(); }
   int pop() {
      int ans = *eles.begin();
```

```
        eles.erase(eles.begin());
        return ans;
    }
    void push(int oi) { eles.insert(oi); }
    int size() const { return eles.size(); }
    void clear() { eles.clear(); }
    int topPrice() { return orders[top()].price; }
    int topSize() {
        int tp = topPrice(), ans = 0;
        for(typename _IntSet::iterator it = eles.begin(); it != eles.end();
it++) {
            const Order& o = orders[*it];
            if(o.price == tp) ans += o.size;
            else break;
        }
        return ans;
    }
};

struct BuyOrderCompare {
    bool operator() (int i, int j) {
        const Order& oi = orders[i];
        const Order& oj = orders[j];
        return oi.price > oj.price || (oi.price == oj.price && i < j);
    }
};

struct SellOrderCompare {
    bool operator() (int i, int j) {
        const Order& oi = orders[i];
        const Order& oj = orders[j];
        return oi.price < oj.price || (oi.price == oj.price && i < j);
    }
};

template<typename T>
ostream& operator<<(ostream& os, const OrderQueue<T> oq) {
    if(oq.eles.empty()) os<<"[]";
    for(typename OrderQueue<T>::_IntSet::iterator it = oq.eles.begin();
it != oq.eles.end(); it++)
        os<<orders[*it];
    return os;
```

```
}

OrderQueue<BuyOrderCompare> buyQueue;
OrderQueue<SellOrderCompare> sellQueue;

void cancel(int ci) {
   int oi = orderIndice[ci];
   if(canceled[oi]) return;
   const Order& o = orders[oi];
   if(o.buy) buyQueue.erase(oi);
   else sellQueue.erase(oi);
   canceled[oi] = 1;
}

void trade(int oi) {
   Order& o = orders[oi];
   if(o.buy) {
      if(sellQueue.empty() || o.price < sellQueue.topPrice()) {
         buyQueue.push(oi);
         return;
      }

      int askPrice;
      while(!sellQueue.empty() && o.size > 0 &&
         o.price >= (askPrice = sellQueue.topPrice())) {
         int toi = sellQueue.top();
         Order& to = orders[toi];
         int tradeSize = min(o.size, to.size);
         cout<<"TRADE "<<tradeSize<<" "<<askPrice<<endl;
         to.size -= tradeSize;
         o.size -= tradeSize;
         sellQueue.pop();
         if(to.size == 0) canceled[toi] = 1;
         else sellQueue.push(toi);
      }

      if(o.size > 0) buyQueue.push(oi);
      else canceled[oi] = 1;
      return;
   }

   if(buyQueue.empty() || o.price > buyQueue.topPrice()) {
```



```
        sellQueue.push(oi);
        return;
    }

    int bidPrice;
    while(!buyQueue.empty() && o.size > 0
        && o.price <= (bidPrice = buyQueue.topPrice())) {
        int toi = buyQueue.top();
        Order& to = orders[toi];

        int tradeSize = min(o.size, to.size);
        cout<<"TRADE "<<tradeSize<<" "<<bidPrice<<endl;
        to.size -= tradeSize;
        o.size -= tradeSize;
        buyQueue.pop();
        if(to.size == 0) canceled[toi] = 1;
        else buyQueue.push(toi);
    }

    if(o.size > 0) sellQueue.push(oi);
    else canceled[oi] = 1;
}

void quote() {
    int bidSize = 0, bidPrice = 0, askSize = 0, askPrice = 99999;

    if(!buyQueue.empty())  {  bidSize  =  buyQueue.topSize();  bidPrice  =
buyQueue.topPrice(); }
    if(!sellQueue.empty())  {  askSize  =  sellQueue.topSize();  askPrice  =
sellQueue.topPrice(); }

    cout<<"QUOTE "<<bidSize<<" "<<bidPrice<<" - "<<askSize<<" "<<askPrice
<<endl;
}

void dbgPrintState() {
    cout<<"Buy Queue: "<<buyQueue<<endl;
    cout<<"Sell Queue: "<<sellQueue<<endl;
}

int main()
{
```

```
    ios::sync_with_stdio(false);
    string cmd;
    bool first = true;
    while(cin>>n) {
        if(first) first = false;
        else cout<<endl;

        fill_n(orderIndice, n, -1);
        fill_n(canceled, n, 0);
        orders.clear();
        buyQueue.clear();
        sellQueue.clear();
        for(int i = 0; i < n; i++) {
            cin>>cmd;
            if(cmd == "CANCEL") {
                int x;
                cin>>x;
                cancel(x-1);
                quote();
                continue;
            }

            Order o;
            cin>>o.size>>o.price;
            o.buy = (cmd == "BUY");
            orderIndice[i] = orders.size();
            orders.push_back(o);
            trade(orderIndice[i]);
            quote();
        }
    }

    return 0;
}
```

习题 5-15　Fibonacci 的复仇（Revenge of Fibonacci, ACM/ICPC Shanghai 2011, UVa12333）

Fibonacci 数的定义为 F(0)=F(1)=1，然后从 F(2)开始，F(i)=F(i−1)+F(i−2)。例如前 10 项 Fibonacci 数分别为 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…

有一天晚上，你梦到了 Fibonacci，它告诉你一个有趣的 Fibonacci 数。醒来以后，你只记得它的开头几个数字。你的任务是找出以它开头的最小 Fibonacci 数的序号。例如以 12 开头的最小 Fibonacci 数是 F(25)。输入不超过 40 个数字，输出满足条件的序号。

如果序号为 0～100000（不包含 100000）的 Fibonacci 数均不满足条件，输出-1。

提示：

本题有一定效率要求。如果高精度代码比较慢，可能会超时。

【分析】

如果直接用十进制大整数类，计算就会超时。可以使用一万进制的大整数类。依次求出前 99999 个 F 数，但是不需要每次保存 F 数本身，只是把其前 40 位作为字符串保留下来，存储到一个 Trie 中去。后续查找就在这个 Trie 中查找。

关于 Trie 的介绍请参考《算法竞赛入门经典——训练指南》中的 3.3.1 节。完整程序如下：

```
using namespace std;

const int BASE = 10000;
template<int MaxNode, int Sigma_Size>
struct Trie {
   int ch[MaxNode][Sigma_Size], sz, val[MaxNode];
   Trie() {
      sz = 1;
      memset(ch[0], 0, sizeof(ch[0]));
      memset(val, 0, sizeof(val));
   }

   int idx(char c) const { return c - '0'; }

   void insert(const string& s, int v) {
      assert(v != 0);
      int u = 0, n = s.size();
      for(int i = 0; i < n; i++) {
         int c = idx(s[i]);
         if(!ch[u][c]) {
            memset(ch[sz], 0, sizeof(ch[sz]));
            val[sz] = v;
            ch[u][c] = sz++;
         }
         u = ch[u][c];
      }

      if(!val[u]) val[u] = v;
      //cout<<"insert - "<<s<<",u = "<<u<<",v = "<<val[u]<<endl;
   }
```

```
    int getValue(const string& s) const {
        int v = -1, u = 0, n = s.size();
        for(int i = 0; i < n; i++) {
            int c = idx(s[i]);
            if(!ch[u][c]) return v;
            u = ch[u][c];
        }

        //cout<<"getValue "<<s<<",u = "<<u<<",v = "<<val[u]<<endl;
        if(val[u]) v = val[u];
        return v;
    }
};

template<int maxn>
struct bign{
    int len, s[maxn];
    bign() { memset(s, 0, sizeof(s)); len = 1; }
    bign(int num) { *this = num; }
    bign& operator=(int num) {
        assert(num >= 0);
        if(num == 0) { len = 1; return *this; }
        len = 0;
        while(num > 0) {
            //printf("s[%d] = %d\n", len, num%BASE);
            s[len++] = num % BASE;
            num /= BASE;
        }
        return *this;
    }

    bign& operator=(const bign& rhs){
        len = rhs.len;
        copy(rhs.s, rhs.s + rhs.len, s);
        return *this;
    }
};

const int MAXF = 100000, MAXLEN = 5300;
typedef bign<MAXLEN> BigFn;
```

```
inline void Add(const BigFn& a, const BigFn& b, BigFn& c) {
    int *cs = c.s, l = 0;
    int mLen = max(a.len, b.len);
    for(int i = 0, g = 0; g || i < mLen; i++) {
        int x = g;
        if(i < a.len) x += a.s[i];
        if(i < b.len) x += b.s[i];
        cs[l++] = x % BASE;
        g = x / BASE;
    }
    c.len = l;
}

ostream& operator<<(ostream &os, const BigFn& x) {
    char buf[8];
    stringstream ss;
    bool first = true;
    for(int i = x.len - 1; i >= 0; i--){
        if(first) {
            first = false;
            ss<<x.s[i];
        } else {
            sprintf(buf, "%04d", x.s[i]);
            ss<<buf;
        }
    }

    const string& s = ss.str();
    if(s.empty()) return os<<0;
    return os<<s;
}

string getPfx(const BigFn& f, int len = 41) {
    int ol = 0;
    char buf[8];
    stringstream ss;
    bool first = true;
    for(int i = 0; i < f.len; i++) {
        if(first) { first = false; sprintf(buf, "%d", f.s[f.len-i-1]); }
        else sprintf(buf, "%04d", f.s[f.len-i-1]);
        ss<<buf;
        ol += strlen(buf);
```

```
        if(ol >= len) break;
    }
    return ss.str();
}

BigFn f0 = 1, f1 = 1, f;
Trie<3817223, 10> pfxes;

int main()
{
    for(int i = 2; i < MAXF; i++) {
        Add(f0, f1, f);
        string pfx = getPfx(f);
        pfxes.insert(pfx, i);
        f0 = f1;
        f1 = f;
    }

    int T;
    scanf("%d", &T);
    char buf[64];
    for(int t = 1; t <= T; t++) {
        scanf("%s", buf);
        int ans = 0;
        string p(buf);
        if(p != "1") ans = pfxes.getValue(p);
        printf("Case #%d: %d\n", t, ans);
    }

    return 0;
}
```

习题 5-16　医院设备利用（Use of Hospital Facilities, ACM/ICPC World Finals 1991, UVa212）

医院里有 n（n≤10）个手术室和 m（m≤30）个恢复室。每个病人首先会被分配到一个手术室，手术后会被分配到一个恢复室。从任意手术室到任意恢复室的时间均为 t_1，准备一个手术室和恢复室的时间分别为 t_2 和 t_3（一开始所有手术室和恢复室均准备好，只有接待完一个病人之后才需要为下一个病人准备）。

k 名（k≤100）病人按照花名册顺序排队，T 点钟准时开放手术室。每当有准备好的手术室时，队首病人进入其中编号最小的手术室。手术结束后，病人应立刻进入编号最小的恢复室。如果有多个病人同时结束手术，编号较小的病人优先进入编号较小的恢复室。输入保证病人无须排队等待恢复室。

输入 n、m、T、t_1、t_2、t_3、k 和 k 名病人的名字、手术时间和恢复时间，模拟这个过程。输入输出细节请参考原题。

提示：

虽然是个模拟题，但是最好先理清思路，减少不必要的麻烦。本题是一个很好的编程练习，但难度也不小。

【分析】

首先是定义事件结构：

```
struct Event {
   int time, id, type;
   Event(int t, int id, int type) : time(t), id(id), type(type) {}
   bool operator<(const Event& e) const { return time > e.time; }
};
```

类似于习题 5-13，可以定义以下几个事件类型：

（1）手术室开始进入空闲状态 opFree。

（2）手术室开始进入准备状态 opPre。

（3）恢复室开始进入空闲状态 reFree。

（4）恢复室开始进入准备状态 rePre。

建立几个全局状态：

（1）等待做手术的病人队列 opQueue。

（2）等待进恢复室的病人队列 reQueue。

（3）空闲手术室队列 freeOpRooms。

（4）空闲恢复室队列 freeReRooms。

（5）事件队列(使用 priority_queue)。

以上 1～4 全局状态均使用 set，因为需要针对编号进行排序，需要注意的是 reQueue 的排序规则，是按照病人之前做手术时所在的手术室编号进行排序。

整个模拟的过程就是循环地进行事件的处理。

1. 处理当前时间的事件

（1）对于 opFree，就是将手术室插入 freeOpRooms。

（2）对于 opPre，说明手术室开始准备，病人做完手术了，需要把这个病人插入 reQueue，并且往事件队列中插入一个手术室准备完毕也就是一个 opFree 的事件。

（3）对于 reFree，就是将恢复室插入 freeReRooms。

（4）对于 rePre，说明病人做完恢复了。往事件队列中插入一个恢复室准备完毕的事件。

2. 分别处理 opQueue 和 reQueue

（1）对于 opQueue 和 freeOpRooms，按照题目规则把病人安排到对应的手术室，并且按照病人的手术时间插入 opPre 事件。同时增加病人以及手术室的对应时间信息。

（2）对于 reQueue 和 freeReRooms，按照题目规则把病人安排到对应的恢复室，并且

按照病人的恢复时间插入 rePre 事件。同时增加病人以及恢复室的对应时间信息。

完整程序如下：

```
using namespace std;

enum eventType { opFree = 0, opPre = 1, reFree = 3, rePre = 4 };

struct Room {
   int pat, minutes;
   void init() { pat = -1; minutes = 0; }
};
struct Event {
   int time, id, type;
   Event(int t, int id, int type) : time(t), id(id), type(type) {}
   bool operator<(const Event& e) const { return time > e.time; }
};
struct Patient {
   string name;
   int  surgeryTime,  recoveryTime,  opRoomId,  opBeginTime,  opEndTime,
reRoomId, reBeginTime, reEndTime;
};

priority_queue<Event> em;
int nOp, nRe, T0, tTrans, tPreOp, tPreRe, nPat, allTime;
Room opRooms[10+1], reRooms[30+1];
Patient pats[100+1];
struct patComp {
   bool operator() (int i1, int i2) {
      const Patient& p1 = pats[i1];
      const Patient& p2 = pats[i2];
      assert(p1.opRoomId != -1 && p2.opRoomId != -1);
      return p1.opRoomId < p2.opRoomId;
   }
};

set<int> opQueue, freeOpRooms, freeReRooms;
set<int, patComp> reQueue;
typedef set<int>::iterator siit;

void writeTime(char* buf, int time) {
   int h = time / 60 + T0, m = time % 60;
   sprintf(buf, "%2d:%02d", h, m);
}
```

```
ostream& operator<<(ostream& os, const Patient& p) {
    char buf[16];
    sprintf(buf, "  %-10s%2d   ", p.name.c_str(), p.opRoomId+1); os<<buf;
    writeTime(buf, p.opBeginTime); os<<buf<<"   ";
    writeTime(buf, p.opEndTime); os<<buf;
    sprintf(buf, "%7d", p.reRoomId+1); os<<buf<<"   ";
    writeTime(buf, p.reBeginTime); os<<buf<<"   ";
    writeTime(buf, p.reEndTime); os<<buf;
    return os;
}

ostream& operator<<(ostream& os, const Room& r) {
    double p = r.minutes * 100;
    p /= allTime;
    char buf[64];
    sprintf(buf, "%8d   %5.2lf", r.minutes, p);
    return os<<buf;
}

void solve() {
    int time = em.top().time;
    while(!em.empty() && em.top().time == time) {
        Event e = em.top();
        em.pop();
         int pid;
        switch(e.type) { //What to do from now
            case opFree:
                assert(!freeOpRooms.count(e.id));
                freeOpRooms.insert(e.id);
                assert(opRooms[e.id].pat == -1);
                break;
            case opPre: //opRoom start pre
                assert(!freeOpRooms.count(e.id));
                pid = opRooms[e.id].pat;
                assert(pid != -1);
                reQueue.insert(pid);
                opRooms[e.id].pat = -1;
                em.push(Event(time + tPreOp, e.id, opFree));
                break;
            case reFree:
                assert(!freeReRooms.count(e.id));
```

```
                assert(reRooms[e.id].pat == -1);
                  freeReRooms.insert(e.id);
                break;
            case rePre:              //reRoom start pre
                assert(!freeReRooms.count(e.id));
                assert(reRooms[e.id].pat != -1);
                reRooms[e.id].pat = -1;
                em.push(Event(time + tPreRe, e.id, reFree));
                break;
            default:
                assert(false);
        }
    }

    int opSz = min(opQueue.size(), freeOpRooms.size());
    for(int i = 0; i < opSz; i++) {
        int pid = *(opQueue.begin());
        opQueue.erase(opQueue.begin());
        int rid = *(freeOpRooms.begin());
        freeOpRooms.erase(freeOpRooms.begin());

        Room& r = opRooms[rid];
        r.pat = pid;
        Patient& p = pats[pid];
        p.opRoomId = rid;
        p.opBeginTime = time;
        p.opEndTime = time + p.surgeryTime;
        r.minutes += p.surgeryTime;
        em.push(Event(p.opEndTime, rid, opPre));
    }

    int reSz = min(reQueue.size(), freeReRooms.size());
    for(int i = 0; i < reSz; i++) {
        int pid = *(reQueue.begin()); reQueue.erase(reQueue.begin());
        int rid = *(freeReRooms.begin()); freeReRooms.erase (freeReRooms.
begin());
        Room& r = reRooms[rid];
        r.pat = pid;
        Patient& p = pats[pid];
        p.reRoomId = rid;
        p.reBeginTime = time + tTrans;
        p.reEndTime = p.reBeginTime + p.recoveryTime;
```



```
        r.minutes += p.recoveryTime;
        em.push(Event(p.reEndTime, rid, rePre));
        allTime = max(allTime, p.reEndTime);
    }
}

int main()
{
    while(cin>>nOp) {
        assert(opQueue.empty());
        assert(reQueue.empty());
        assert(em.empty());

        freeOpRooms.clear();
        freeReRooms.clear();
        allTime = 0;

        cin>>nRe>>T0>>tTrans>>tPreOp>>tPreRe>>nPat;
        for(int i = 0; i < nOp; i++) {
            em.push(Event(0, i, opFree));
            opRooms[i].init();
        }
        for(int i = 0; i < nRe; i++) {
            em.push(Event(0, i, reFree));
            reRooms[i].init();
        }

        for(int i = 0; i < nPat; i++) {
            Patient& p = pats[i];
            p.opRoomId = -1;
            p.reRoomId = -1;
            cin>>p.name>>p.surgeryTime>>p.recoveryTime;
            opQueue.insert(i);
        }

        while(!em.empty()) {
            solve();
        }
        cout<<" Patient  Operating Room           Recovery Room"<<endl;
        cout<<" #  Name    Room#  Begin   End       Bed#  Begin    End"<<endl;
        cout<<" ---------------------------------------------------"<<endl;
```

```
        for(int i = 0; i < nPat; i++) cout<<setw(2)<<i+1<<pats[i]<<endl;
        cout<<endl;
        cout<<"Facility Utilization"<<endl;
        cout<<"Type  # Minutes  % Used"<<endl;
        cout<<"-------------------------"<<endl;
        for(int i = 0; i < nOp; i++) cout<<"Room "<<setw(2)<<i+1<<opRooms
[i]<<endl;
        for(int i = 0; i < nRe; i++) cout<<"Bed  "<<setw(2)<<i+1<<reRooms
[i]<<endl;
        cout<<endl;
    }
    return 0;
}
```

2.4 数据结构基础

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 6 章。

习题 6-1　平衡的括号（Parentheses Balance, UVa673）

输入一个包含“()”和“[]”的括号序列，判断是否合法。具体规则如下：

（1）空串合法。

（2）如果 A 和 B 都合法，则 AB 合法。

（3）如果 A 合法，则(A)和[A]都合法。

【分析】

对输入的序列进行遍历，同时用一个栈维护当前遍历过但是未配对的括号，栈就用 STL 的 stack。

（1）遍历到右括号，如果栈为空，说明无法匹配到左括号，直接退出。如果栈不为空，需与最近的左括号也就是栈顶的元素匹配。如果匹配直接出栈，继续处理序列中的下一个元素。否则直接退出。

（2）遇到左括号，入栈。

（3）遍历完之后如果栈不为空，则说明有无法匹配的左括号，序列非法。

完整程序如下：

```
using namespace std;

bool isCorrect(const char* s) {
    int len = strlen(s);
    stack<char> st;
    for(int i = 0; i < len; i++)
```

```
    {
        char c = s[i];
        if(c == '(' || c == '[') st.push(c);
        else
        {
            assert(c == ')' || c == ']');
            if(st.empty()) return false;
            char t = st.top();
            if(c == ')') {
                if(t == '(') st.pop();
                else break;
            }
            else if(c == ']') {
                if(t == '[') st.pop();
                else break;
            }
        }
    }

    return st.empty();
}

int main()
{
    int n;
    char buf[256];
    scanf("%d\n", &n);
    while(n--) {
        gets(buf);
        if(isCorrect(buf)) puts("Yes");
        else puts("No");
    }
    return 0;
}
```

习题 6-2　S 树（S-Trees, UVa712）

给一棵满二叉树，每一层代表一个 01 变量，取 0 时往左走，取 1 时往右走。例如图 2.19 中两个图都对应表达式 $x_1 \wedge (x_2 \vee x_3)$。

给出所有叶子的值以及一些查询（即每个变量 x_i 的取值），求每个查询到达的叶子的值。例如有 4 个查询：000, 010, 111, 110，则输出应为 0011。

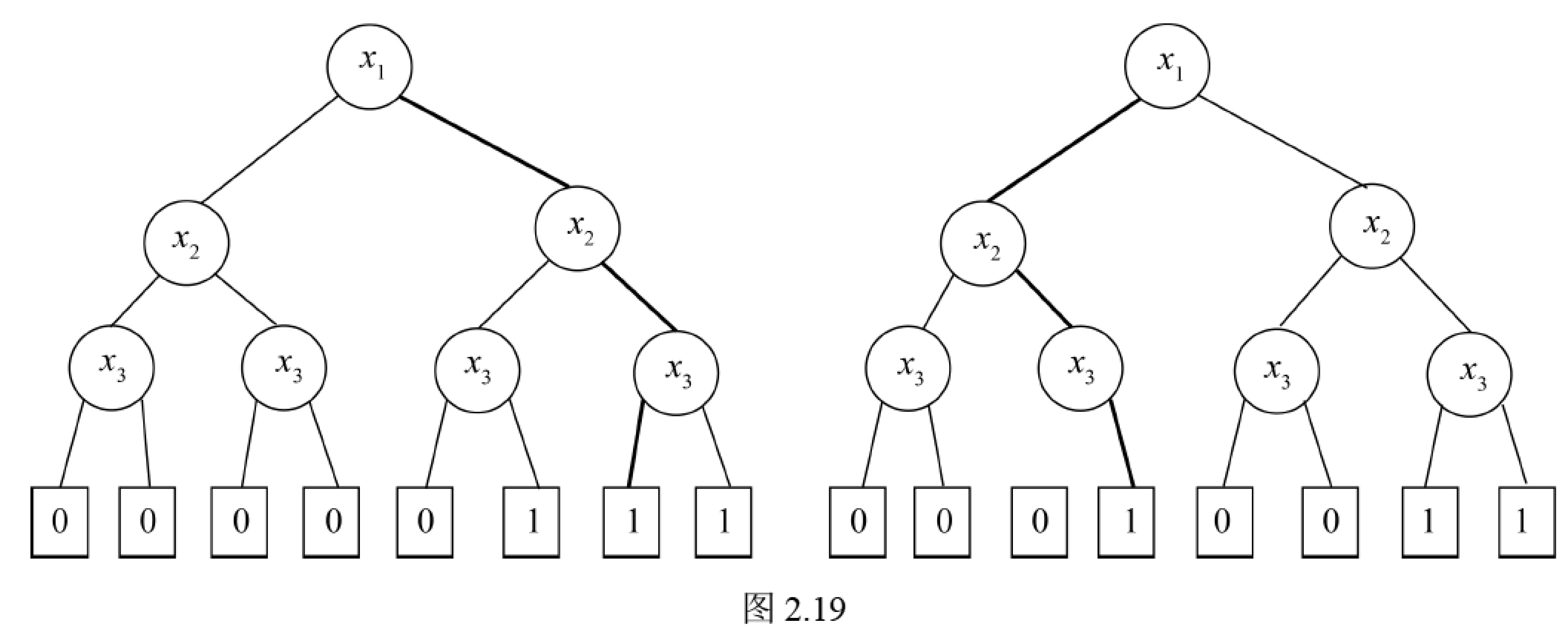


图 2.19

【分析】

仔细观察可以发现，可以不用建立二叉树的结构，每遍历一个输入的字符，就沿树下降一层，目标叶子所在的区间就缩小一半，当处理完所有的 x_i 之后，刚好到达叶子，区间缩小为只有一个元素。注意叶子结点的个数是 2^n 而不是 n。

完整程序如下：

```
using namespace std;
const int MAXN = 8, MAXNODE = 1<<MAXN;
int n, leavesCnt, order[MAXN], leaves[MAXNODE];
char buf[MAXNODE];

int solve(const char* vva){
    int l = 0, r = leavesCnt-1;
    _for(i, 0, n){
        int o = order[i], m = (l+r)/2;
        if(vva[o] == '1') l = m + 1;
        else r = m;
    }

    assert(l == r);
    return leaves[l];
}

int main(){
    int t = 1, x, m;
    while(scanf("%d", &n) == 1 && n){
        leavesCnt = 1<<n;
        _for(i, 0, n){
            scanf("%s", buf);
            sscanf(buf+1, "%d", &x);
            order[i] = x - 1;
```

```
        }
        scanf("%s", buf);
        _for(i, 0, leavesCnt) leaves[i] = buf[i] - '0';

        scanf("%d", &m);
        printf("S-Tree #%d:\n", t++);
        _for(i, 0, m){
            scanf("%s", buf);
            printf("%d", solve(buf));
        }
        puts("\n");
    }
    return 0;
}
```

习题 6-3 二叉树重建（Tree Recovery, ULM 1997, UVa536）

输入一棵二叉树的先序遍历和中序遍历序列，输出它的后序遍历序列，如图 2.20 所示。

样例输入	样例输出
DBACEGF ABCDEFG BCAD CBAD	ACBFGED CDAB

图 2.20

【分析】

具体思路可以参考《算法竞赛入门经典（第 2 版）》6.3.3 节的二叉树重建。

习题 6-4 骑士的移动（Knight Moves, UVa439）

输入标准 8*8 国际象棋棋盘上的两个格子（列用 a～h 表示，行用 1～8 表示），求马最少需要多少步从起点跳到终点。例如从 a1 到 b2 需要 4 步。马的移动方式如图 2.21 所示。

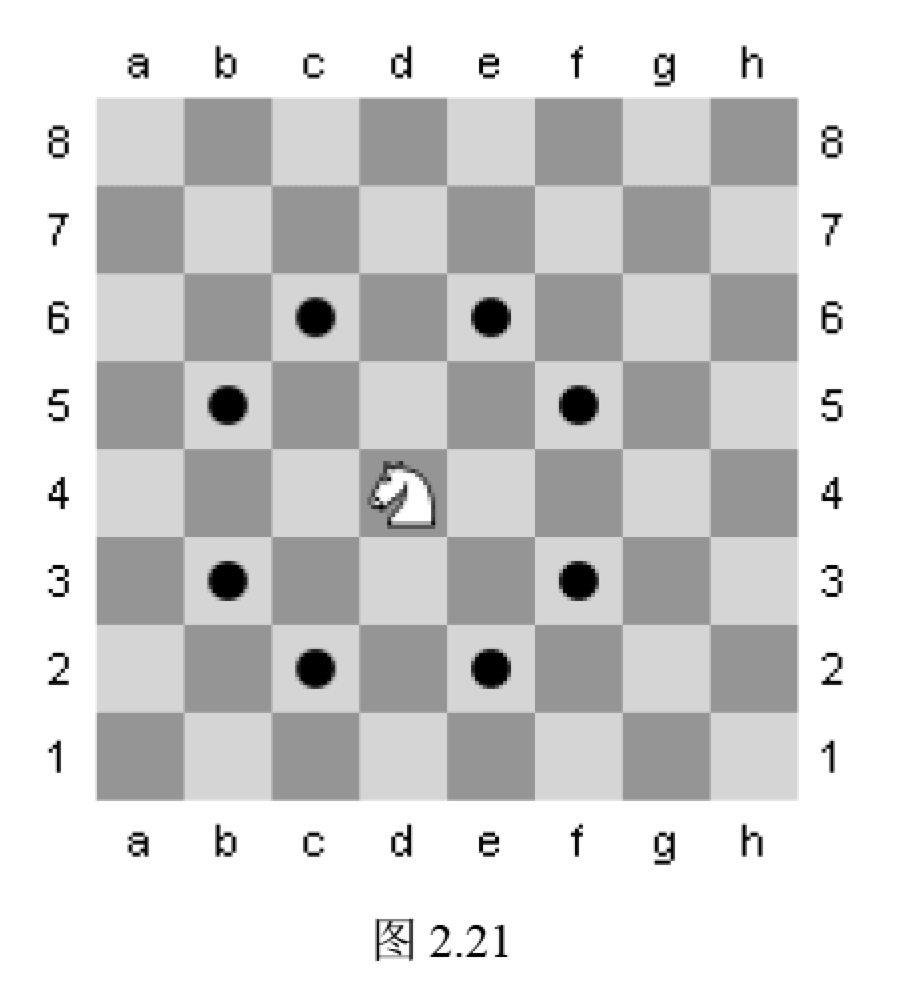


图 2.21

【分析】

使用 Point 类来存储位置，然后再存储向 8 个方向跳对应的 8 个向量，在 BFS 搜索位置遍历 8 个方向时就可以直接用向量计算来简化代码。而且在比较点坐标时也比较方便，具体请参考代码。完整程序（C++11）如下：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
using namespace std;

struct Point {
   int x, y;
   Point(int x = 0, int y = 0): x(x), y(y) {}
};
typedef Point Vector;

Vector operator+ (const Vector &A, const Vector &B) { return Vector(A.x +
B.x, A.y + B.y); }
bool operator== (const Point &a, const Point &b) { return a.x == b.x && a.y
== b.y; }
bool inRange(int x, int left, int right) {
   if(left > right) return inRange(x, right, left);
   return left <= x && x <= right;
}

const int N = 8;
Point toPoint(const char* ps) { return Point(ps[0] - 'a', ps[1] - '1'); }
bool isValid(const Point &p) { return inRange(p.x, 0, N-1) && inRange(p.y,
0, N-1); }
Vector dirVs[N] = {{2, 1}, {1, 2}, { -1, 2}, { -2, 1}, { -2, -1}, { -1, -2},
{1, -2}, {2, -1}};

int solve(const Point &from, const Point &to) {
   int vis[N][N];
   memset(vis, -1, sizeof(vis));
   queue<Point> q;
   q.push(from);
   vis[from.x][from.y] = 0;
   while(!q.empty()) {
      const Point &f = q.front(); q.pop();
      int d = vis[f.x][f.y];
      if(f == to) return d;
      _for(i, 0, N) {
         Point np = f + dirVs[i];
```



```
            if(isValid(np) && vis[np.x][np.y] == -1) {
                vis[np.x][np.y] = d + 1;
                q.push(np);
            }
        }
    }
    assert(false);
}

int main() {
    char a[16], b[16];
    while(scanf("%s%s", a, b) == 2){
        int ans = solve(toPoint(a), toPoint(b));
        printf("To get from %s to %s takes %d knight moves.\n", a, b, ans);
    }
}
```

习题 6-5 巡逻机器人（Patrol Robot, ACM/ICPC Hanoi 2006, UVa1600）

机器人要从一个 $m*n$（$1 \leqslant m, n \leqslant 20$）的网格的左上角(1,1)走到右下角($m,n$)。网格中的一些格子是空地（用 0 表示），其他格子是障碍（用 1 表示）。机器人每次可以往 4 个方向走一格，但不能连续地穿越 k（$0 \leqslant k \leqslant 20$）个障碍，求最短路长度。起点和终点保证是空地。例如对于图 2.22 所示左图的数据，图 2.22 的右图是最优解，路径长度为 10。

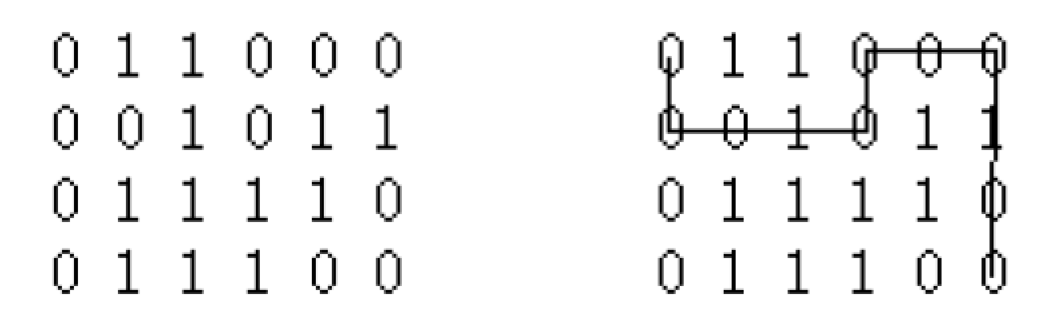


图 2.22

【分析】

首先，与习题 6-3 一样，同样可以使用 Point 类来简化处理逻辑。而当前状态除了包含当前坐标，还要包含已经穿越的障碍个数，这一点是和上题的关键区别，BFS 即可。完整程序如下：

```
using namespace std;

int readint() { int x; scanf("%d", &x); return x;}
bool inRange(int x, int l, int r) { return (l > r) ? inRange(x, r, l) : (l
<= x && x <= r); }

const int MAXN = 24;
int M, N, K, Grid[MAXN][MAXN], vis[MAXN][MAXN][MAXN];
```

```
    bool isValid(const Point& p) { return inRange(p.x, 0, M-1) && inRange(p.y,
0, N-1); }
    Vector dirVs[4];

    struct Stat{
       Point pos;
       int turbo;
    };

    int& getVisd(const Stat& s) { return vis[s.pos.x][s.pos.y][s.turbo]; }

    int solve() {
       Stat s;
       Point dest(M-1, N-1);
       s.pos.x = 0; s.pos.y = 0; s.turbo = 0;
       memset(vis, -1, sizeof(vis));

       queue<Stat> q;
       q.push(s);
       vis[s.pos.x][s.pos.y][s.turbo] = 0;

       while(!q.empty()) {
          const Stat& f = q.front();
          q.pop();
          const int& fd = getVisd(f);
          if(f.pos == dest) return fd;
          assert(f.turbo <= K);

          for(int i = 0; i < 4; i++) {
             Point np = f.pos + dirVs[i];
             if(!isValid(np)) continue;
             int isBlock = Grid[np.x][np.y];
             if(isBlock && f.turbo + 1 > K) continue;

             Stat ns;
             ns.pos = np;
             ns.turbo = isBlock ? (f.turbo + 1) : 0;

             int& d = getVisd(ns);
             if(d == -1) { d = fd+1; q.push(ns); }
          }
       }
```



```
    return -1;
}

int main(){
    dirVs[0].x = 1; dirVs[0].y = 0;     //e
    dirVs[1].y = 1; dirVs[1].x = 0;     //n
    dirVs[2].y = -1; dirVs[2].x = 0;    //s
    dirVs[3].x = -1; dirVs[3].y = 0;    //w

    int T = readint();
    while(T--) {
        M = readint(), N = readint(); K = readint();
        for(int i = 0; i < M; i++)
            for(int j = 0; j < N; j++)
                Grid[i][j] = readint();

        int ans = solve();
        printf("%d\n", ans);
    }
}
```

习题 6-6 修改天平（Equilibrium Mobile, NWERC 2008, UVa12166）

用一个深度不超过 16 的二叉树，代表一个天平。每根杆都悬挂在中间，每个秤砣的重量已知。至少修改多少个秤砣的重量才能让天平平衡？如图 2.23 所示，把 7 改成 3 即可。

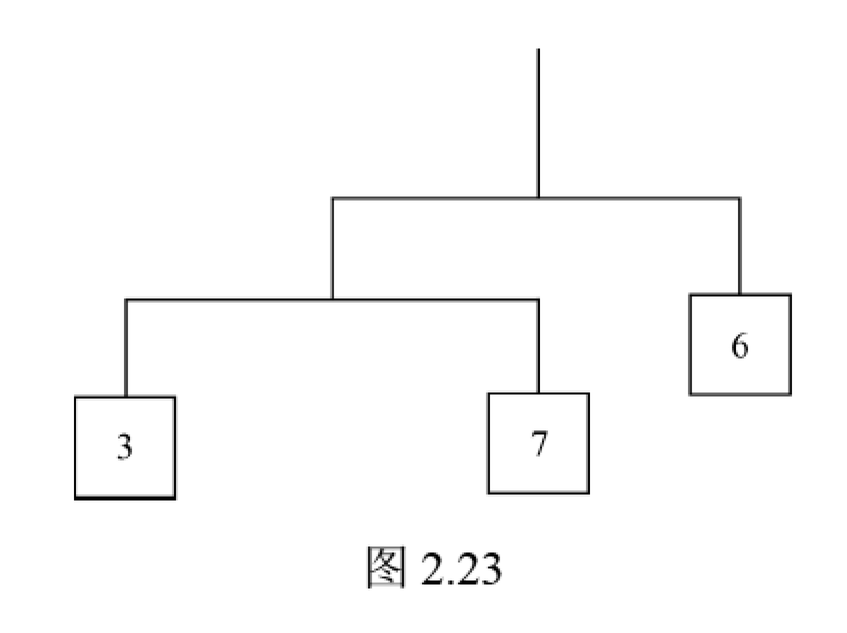


图 2.23

【分析】

树在最终平衡之后，只要确定单个结点的重量，整个树的重量就可以确定。此时深度 n（根结点深度为 0）的子树重量为深度 $n-1$ 子树重量的 1/2。如果一个深度 n 的秤砣结点重量为 x，则整棵树的重量就是 $x*2^n$。要求计算出需修改重量的秤砣的最小个数，反过来就是计算不需要修改的秤砣的最大个数。对于第 n 层的重量为 x 结点，假设它不需要修改，则最终平衡的树的重量就是 $T=x*2^n$。遍历每个结点，计算出现次数最多的那个 T 的出现次数 K，则（结点的个数$-K$）即是所求的答案。完整程序如下：

```
using namespace std;
const int MAXN = 24;
typedef long long LL;

int main()
```

```
{
    int T; cin>>T;
    string line;
    map<LL, int> vCnt;
    while(T--) {
        cin>>line;
        vCnt.clear();
        int sz = line.size(), depth = 0, nodeCnt = 0;
        for(int i = 0; i < sz; i++) {
            char c = line[i];
            if(c == '[') depth++;
            else if(c == ']') depth--;
            else if(isdigit(c)) {
                LL v = c - '0';
                int j;
                for(j = i + 1; j < sz && isdigit(line[j]); j++) {
                    v *= 10;
                    v += line[j] - '0';
                }
                i = j-1;
                v <<= depth;
                vCnt[v] = vCnt[v] + 1;
                nodeCnt++;
            }
        }

        int K = -1;
        for(const auto& p : vCnt) K = max(K, p.second);
        assert(K > 0);
        cout<<(nodeCnt-K)<<endl;
    }
}
```

习题 6-7　Petri 网模拟（Petri Net Simulation, ACM/ICPC World Finals 1998, UVa804）

你的任务是模拟 Petri 网的变迁。Petri 网包含 NP 个库所（用 P1，P2…表示）和 NT 个变迁（用 T1, T2…表示）。0<NP，NT<100。当每个变迁的每个输入库所都至少有一个 token 时，变迁是允许的。变迁发生的结果是每个输入库所减少一个 token，每个输出库所增加一个 token。变迁的发生是原子性的，即所有 token 的增加和减少应同时进行。注意，一个变迁可能有多个相同的输入或者输出。如果有多个变迁是允许的，一次只能发生一个。

如图 2.24 所示，一开始只有 T1 是允许的，发生一次 T1 变迁之后有一个 token 会从 P1 移动到 P2，但仍然只有 T1 是允许的，因为 2 要求 P2 有两个 token。再发生一次 T1 变迁之



后 P1 中只剩一个 token，而 P2 中有两个，因此 T1 和 T2 都可以发生。假定 T2 发生，则 P2 中不再有 token，而 P3 中有一个 token，因此 T1 和 T3 都是允许的。

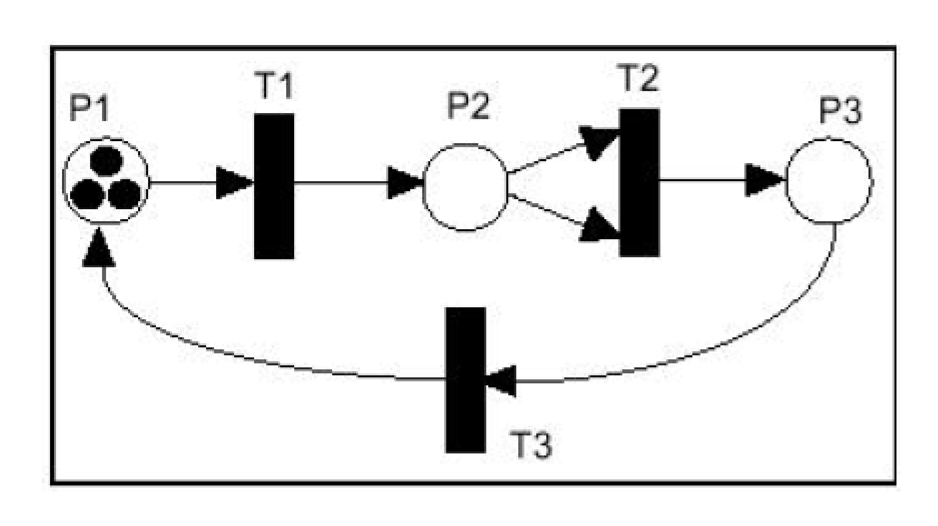


图 2.24

输入一个 Petri 网络。初始时每个库所都有一个 token。每个变迁用一个整数序列表示，负数表示输入库所，正数表示输出库所。每个变迁至少包含一个输入和一个输出。最后输入一个整数 NF，表示要发生 NF 次变迁（同时有多个变迁允许时可以任选一个发生，输入保证这个选择不会影响最终结果）。

【分析】

首先建立一个结构 Transition 表示一个变迁，包含所有的输入库所编号及其出现次数，使用 map<int,int>来表示，例如题图中 T2 中 P2 出现两次。另外用一个 vector<int>存储所有的输出 Place 编号。这个 Transition 只有在每个编号为 i 的输入库所中的 Token 个数≥ i 在出现的次数时才被允许。

每一次变迁时，对于输入库所 i 来说，Token 个数要减掉其在 Transition 的输入中出现的次数，然后所有的输出库所的 Token 增加 1。完整程序（C++11）如下：

```
using namespace std;

int readint(){int x; cin>>x; return x;}
vector<int> places;

struct Transition {
   vector<int> output;
   map<int, int> input;
   bool enabled() const {
      for(auto &p : input) if(places[p.first] < p.second) return false;
      return true;
   }

   void init() { input.clear(), output.clear(); }

   void op() {
      int flow = output.size();
      for(auto &p : input) places[p.first] -= p.second;
      for(auto o : output) places[o]++;
```

```
    }
};

istream& operator>>(istream& is, Transition& t) {
    int x;
    t.init();
    while(is>>x && x) {
        if(x < 0) t.input[-x-1]++;
        else t.output.push_back(x-1);
    }
    return is;
}

vector<Transition> ts;
int main()
{
    int NP;
    for(int kase = 1; cin>>NP && NP; kase++) {
        places.clear(); ts.clear();
        _for(i, 0, NP) places.push_back(readint());
        int NT = readint();
        Transition t;
        _for(i, 0, NT) {
            cin>>t; ts.push_back(t);
        }

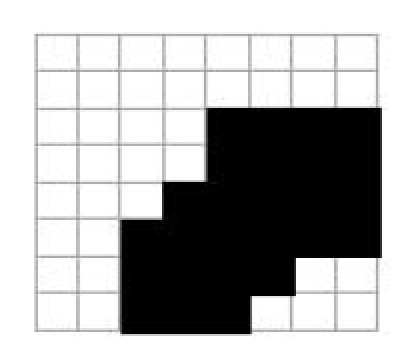
        bool live = true; int cnt = 0;
        int NF = readint();
        _for(i, 0, NF) {
            auto pt = find_if(ts.begin(), ts.end(),
                [](const Transition& t){ return t.enabled(); });
            live = pt != ts.end();
            if(!live) break;
            pt->op();
            cnt++;
        }

        cout<<"Case "<<kase<<": ";
        if(live) cout<<"still live after ";
        else cout<<"dead after ";
        cout<<cnt<<" transitions\nPlaces with tokens:";
        _for(i, 0, places.size()) {
```

```
            int t = places[i];
            if(t) cout<<" "<<i+1<<" ("<<t<<')';
        }
        cout<<endl<<endl;
    }
    return 0;
}
```

习题 6-8　空间结构（Spatial Structures, ACM/ICPC World Finals 1998, UVa806）

黑白图像有两种表示法：点阵表示和路径表示。路径表示法首先需要把图像转换为四分树，然后记录所有黑结点到根的路径。例如对于如图 2.25 所示的图像。



```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 1 1 1 1 1
0 0 1 1 1 1 1 1
0 0 1 1 1 1 0 0
0 0 1 1 1 0 0 0
```

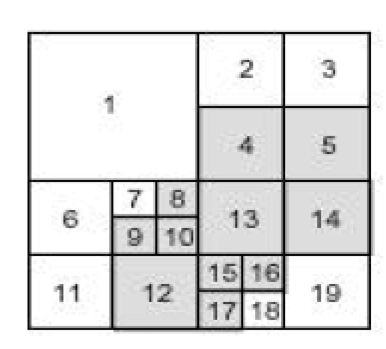


图 2.25

四分树如图 2.26 所示。

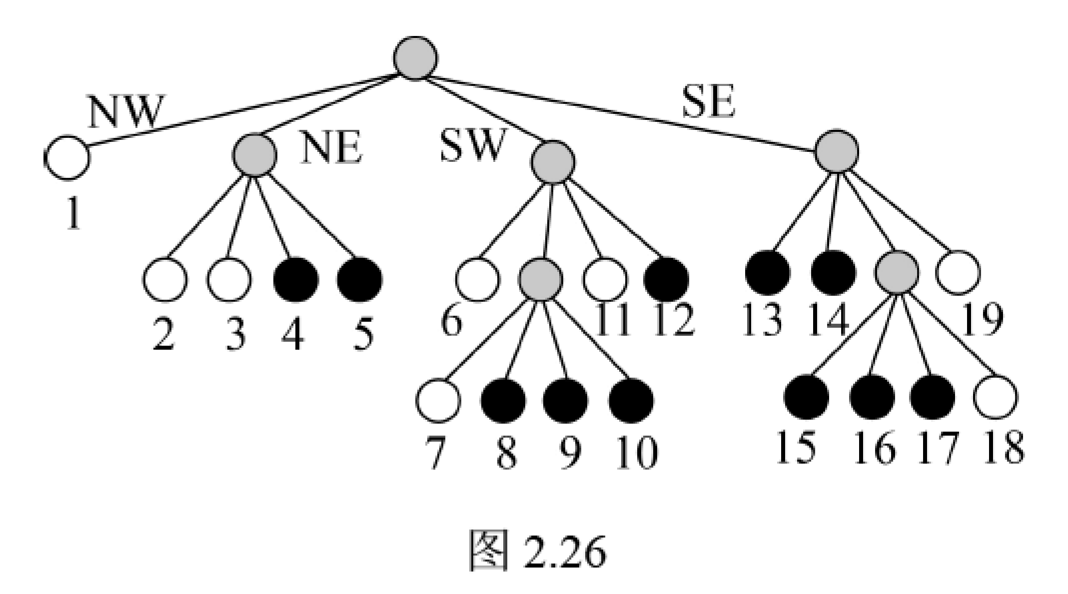


图 2.26

NW、NE、SW、SE 分别用 1、2、3、4 表示。最后把得到的数字串看成是五进制的，转换为十进制后排序。例如上面的树在转换、排序后的结果是 9 14 17 22 23 44 63 69 88 94 113。

你的任务是在这两种表示法之间进行转换。在点阵表示法中，1 表示黑色，0 表示白色。图像总是正方形的，且长度 n 为 2 的整数幂，并满足 $n \leqslant 64$。输入输出细节请参见原题。

【分析】

因为题目实际上是要求在四分树的两种表示形式之间转换，那么很容易想到的就是要建立四分树的结构。

```
struct QNode {
    int color;
    QNode *Nodes[4];
    void init() { memset(Nodes, 0, sizeof(Nodes)); }
   QNode() { init(); }
};
```

对于输入黑色点的情况，每一个点计算出其在 Grid 中对应的区域，然后给对应的区设置黑色标记即可。对于输入是 Grid 的情况，可以使用递归的方法建立四分树，然后使用 DFS 来搜索到所有的黑结点，并且在沿着树向下搜索的过程中记录相应的路径即可。完整程序如下：

```
using namespace std;
#define _for(i,a,b) for(int i=(a); i<(b); ++i)

const int MAXN = 64 + 1;
enum Color{ White = 0, Black = 1, Gray = 2 };

struct QNode {
    int color;
    QNode *Nodes[4];
    void init() { memset(Nodes, 0, sizeof(Nodes)); }
    QNode() { init(); }
};

MemPool<QNode> pool;
int N, Grid[MAXN][MAXN];
Vector dirVS[4];
bool isGrid;

QNode* treeFromGrid(const Point& p, int len) {
    QNode *pn = pool.createNew();
    if (len == 1) {
        pn->color = Grid[p.x][p.y];
        return pn;
    }

    assert(len % 2 == 0);

    int color = -1;
    for (int i = 0; i < 4; i++) {
        Point pb = p + dirVS[i] * (len / 2);
        pn->Nodes[i] = treeFromGrid(pb, len / 2);
        int c = pn->Nodes[i]->color;
        if (color == -1) color = c;
        else if (color == Gray) continue;
        else if (color != c) color = Gray;
    }
```

```
    if ((pn->color = color) != Gray) pn->init();
    return pn;
}

QNode* treeFromGrid() { return treeFromGrid(Point(0, 0), N); }

void printPath(const QNode *root, vector<int>& route, vector<int>& paths) {
    assert(root);
    if (root->color == White) return;
    if (root->color == Gray) {
        _for (i, 0, 4) {
            route.push_back(i+1); assert(root->Nodes[i]);
            printPath(root->Nodes[i], route, paths);
            route.pop_back();
        }
        return;
    }
    assert(root->color == Black);
    int base = 1, path = 0;
    for(auto r : route) path += base * r, base *= 5;
    paths.push_back(path);
}

void locate(int black, int& len, Point& pos) {
    len = N; pos.x = 0; pos.y = 0;
    while (black) {
        len /= 2; assert(len); assert(black%5);
        pos = pos + dirVS[black%5-1]*len;
       black /= 5;
    }
}

void gridFromBlacks(const vector<int>& blacks) {
   _for(i, 0, N) _for(j, 0, N) Grid[i][j] = 0;
    Point pos; int len;
   for(auto b : blacks) {
      locate(b, len, pos);
      _for(r, 0, len) _for(c, 0, len) Grid[r+pos.x][c+pos.y] = Black;
   }
}

int main() {
```

```
int n;
string line;
for (int i = 0; i < 4; i++) {
    dirVS[i].x = i / 2;
    dirVS[i].y = i % 2;
}

for (int t = 0; cin>>n && n; t++) {
    if (t) cout<<endl;
    isGrid = n > 0; N = abs(n);
    if (isGrid) {
        _for(i, 0, N) {
            cin>>line;
            _for(j, 0, N) Grid[i][j] = line[j] - '0';
        }
    }
    else {
        vector<int> blacks;
      int p;
        while (cin>>p && p != -1) blacks.push_back(p);
        gridFromBlacks(blacks);
    }

   cout<<"Image "<<t+1;
    if (isGrid) {
        QNode *root = treeFromGrid();
        vector<int> route, blacks;
        printPath(root, route, blacks);
        sort(blacks.begin(), blacks.end());
      _for(i, 0, blacks.size()) {
          if(i%12) cout<<" "; else cout<<endl;
          cout<<blacks[i];
      }
        cout<<endl;
      cout<<"Total number of black nodes = "<<blacks.size()<<endl;
    }
    else {
      cout<<endl;
        _for(i, 0, N) {
            _for(j, 0, N) cout<<(Grid[i][j] ? '*' : '.');
            cout<<endl;
        }
```

```
        }
    }

    pool.dispose();
    return 0;
}
```

习题 6-9　纸牌游戏（"Accordian" Patience, UVa127）

把 52 张牌从左到右排好，每张牌自成一个牌堆（pile）。当某张牌与它左边那张牌或者左边第三张牌 match（花色 suit 或者点数 rank 相同）时，就把这张牌移到那张牌上面去。移动之后还要看看是否可以进行其他的移动。只有位于牌堆顶部的牌才能移动或者参与 match。当牌堆之间出现空隙时要立刻把右边的所有牌堆左移一格来填补空隙。如果有多张牌可以移动，先移动最左边的那张牌；如果既可以移一格也可以移 3 格时，移 3 格。按顺序输入 52 张牌，输出最后的牌堆数以及各牌堆的牌数。

样例输入：

```
QD AD 8H 5S 3H 5H TC 4D JH KS 6H 8S JS AC AS 8D 2H QS TS 3S AH 4H TH TD 3C 6S
8C 7D 4C 4S 7S 9H 7C 5D 2S KD 2D QH JD 6D 9D JC 2C KH 3D QC 6C 9S KC 7H 9C 5C
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC AD 2D 3D 4D 5D 6D 7D 8D TD 9D JD QD KD
AH 2H 3H 4H 5H 6H 7H 8H 9H KH 6S QH TH AS 2S 3S 4S 5S JH 7S 8S 9S TS JS QS KS
#
```

样例输出：

```
6 piles remaining: 40 8 1 1 1 1
1 pile remaining: 52
```

【分析】

本题牵涉两种数据结构，一是栈（模拟牌堆），使用 STL 的 stack 即可；而多个牌堆之间实际上是链表的关系（中间牵涉牌堆的删除以及连接），自己定义一个 LinkNode 结构，内部包含一个 stack。同时，加一个头结点方便处理。完整程序（C++11）如下：

```
using namespace std;

ostream& operator<<(ostream& oss, const vector<int>& s) {
    for(vector<int>::const_iterator p = s.begin(); p != s.end(); p++)
        oss<<' '<<*p;
    return oss;
}

const int PileCnt = 52;
struct Card {
    char suit, rank;
    Card(char r, char s) :suit(s), rank(r) {}
```

```
        bool match(const Card& rhs) const { return rank == rhs.rank || suit ==
rhs.suit; }
    };
    struct Pile {
        stack<Card> cards;
        Pile *prev, *next;
        void init(){
            while(!cards.empty()) cards.pop();
            prev = nullptr; next = nullptr;
        }
    };

    Pile piles[1+PileCnt], *head;

    void connect(Pile* p1, Pile* p2) { if(p1) p1->next = p2; if(p2) p2->prev =
p1; }

    Pile* getLeft3(Pile* p) {       //得到左数第三个牌堆
        for(int i = 0; i < 3; i++) {
            p = p->prev;
            if(p == nullptr) return nullptr;
        }
        return p;
    }

    void solve() {
        Pile *from, *to, *cur;
        while (true){
            from = nullptr, to = nullptr;
            cur = head->next;
            while(cur) {
                assert(!cur->cards.empty());
                Pile* l3 = getLeft3(cur);
                if(l3 != nullptr && l3 != head) {
                    assert(!l3->cards.empty());
                    if(l3->cards.top().match(cur->cards.top()))
                    { from = cur; to = l3; break; }
                }
                Pile* l1 = cur->prev;
                if(l1 != head) {
                    assert(!l1->cards.empty());
                    if(l1->cards.top().match(cur->cards.top()))
```

```
                { from = cur; to = l1; break; }
            }
            cur = cur->next;
        }

        if(from) assert(to); else break;
        to->cards.push(from->cards.top());
        from->cards.pop();
        if(from->cards.empty()) connect(from->prev, from->next);
    }
}

int main() {
    string s;
    bool end = false;
    head = &(piles[0]);
    head->init();
    head->next = &(piles[1]);

    while(true) {
        for(int i = 1; i <= PileCnt; i++) {
            if(cin>>s && s.size() == 2) {
                Pile& p = piles[i];
                p.init();
                p.prev = &(piles[i-1]);
                if(i+1 <= PileCnt) p.next = &(piles[i+1]);
                p.cards.push(Card(s[0], s[1]));
            }
            else return 0;
        }
        solve();
        Pile *cur = head->next;
        vector<int> ps;
        while(cur) {
            assert(!cur->cards.empty());
            ps.push_back(cur->cards.size());
            cur=cur->next;
        }
        cout<<ps.size()<<" pile"<<(ps.size() > 1 ? "s":"")<<" remaining:
"<<ps<<endl;
    }
    return 0;
}
```

习题 6-10　10-20-30 游戏（10-20-30, ACM/ICPC World Finals 1996, UVa246）

有一种纸牌游戏叫作 10-20-30。游戏使用除大王和小王之外的 52 张牌，J、Q、K 的面值是 10，A 的面值是 1，其他牌的面值等于它的点数。

把 52 张牌叠放在一起放在手里，然后从最上面开始依次拿出 7 张牌从左到右摆成一条直线放在桌子上，每一张牌代表一个牌堆。每次取出手中最上面的一张牌，从左至右依次放在各个牌堆的最下面。当往最右边的牌堆放一张牌以后，重新往最左边的牌堆上放牌。

如果当某张牌放在某个牌堆上后，牌堆的最上面两张和最下面一张牌的和等于 10、20 或者 30，这 3 张牌将会从牌堆中被拿走，然后按顺序放回手中并压在最下面。如果没有出现这种情况，将会检查最上面一张和最下面两张牌的和是否为 10、20 或者 30，解决方法类似。如果仍然没有出现这种情况，最后检查最下面 3 张牌的和，并用类似的方法处理。例如，如果某一牌堆中的牌从上到下依次是 5、9、7、3，那么放上 6 以后的布局如图 2.27 所示。



图 2.27

如果放的不是 6，而是 Q，对应的情况如图 2.28 所示。



图 2.28

如果某次操作后某牌堆中没有剩下一张牌，那么把该牌堆永远地清除掉，并把它右边的所有牌堆顺次往左移。如果所有牌堆都清除，游戏胜利结束；如果手里没有牌了，游戏以失败告终；有时游戏永远无法结束，这时我们说游戏出现循环。给出 52 张牌最开始在手中的顺序，请模拟这个游戏并计算出游戏结果。

【分析】

这个题目在数据结构的选取方面关键有几点：

（1）牌堆所用的数据结构，因为要在两端进行操作，所以使用 STL 中 deque（双端队

列）最为合适：typedef deque<int> Pile。

（2）要把当前的局面记录下来进行判重，可以将所有的牌堆中的牌编码成一个字符串:每张牌的牌面直接转成 char，牌堆与牌堆之间用“|”之类的分隔符隔开，然后局面就可以用 string 来表示。同时使用一个 set<string>来对局面的编码进行判重。

（3）所有的牌堆因为牵涉有删除以及移位操作，可用 STL 中的双向链表 list<Pile*>来表示。

每一次的模拟过程就是以下几个步骤：

（1）依次判断牌堆和手中的牌是否已经清空，如果清空，直接输出结果。

（2）对当前局面进行编码，然后判重，如果重复直接退出。

（3）从手牌取出一张并且放到左边的牌堆，同时把这个牌堆放到所有牌堆的最后。

（4）针对最后的牌堆进行 10-20-30 的操作。操作完成之后判断牌堆是否已经清空，如果已经被清空，则从链表中删除这个牌堆。

完整程序（C++11）如下：

```
using namespace std;

#define _for(i,a,b) for( int i=(a); i<(b); ++i)
const int CN = 52;
int readint() { int x; cin>>x; return x; }

typedef deque<int> Pile;

Pile cards;
Pile allPiles[7];
list<Pile*> piles;
set<string> phases;

//对整体状态进行编码
void encode(string& ans) {
   ans.clear();
   for(auto& pp : piles) {
      Pile& p = *pp;
      for(auto c : p) ans += (char)c;
      ans += '|';
   }
   for(auto c : cards) ans += (char)c;
}

//10-20-30 操作
void procPile(Pile& p) {
   int n = p.size();
```

```
        if(n < 3) return;
        if ((p[0] + p[1] + p.back()) % 10 == 0) {
          cards.push_back(p[0]), cards.push_back(p[1]), cards.push_back(p.back());
           p.pop_front(), p.pop_front(), p.pop_back();
           procPile(p);
           return;
        }

        if ((p[0] + p[n-2] + p[n-1]) % 10 == 0) {
          cards.push_back(p[0]), cards.push_back(p[n-2]), cards.push_back(p[n-1]);
           p.pop_front(), p.pop_back(), p.pop_back();
           procPile(p);
           return;
        }

        if ((p[n-3] + p[n-2] + p[n-1]) % 10 == 0) {
           cards.push_back(p[n-3]), cards.push_back(p[n-2]), cards.push_back
(p[n-1]);
           p.pop_back(), p.pop_back(), p.pop_back();
           procPile(p);
           return;
        }
    }

    bool simulate(int time) {
        if(piles.empty()) {
           cout<<"Win : "<<time<<endl;
           return false;
        }

        if(cards.empty()) {
           cout<<"Loss: "<<time<<endl;
           return false;
        }

        string pha;
        encode(pha);
        if(phases.count(pha)) {
           cout<<"Draw: "<<time<<endl;
           return false;
        }
        else phases.insert(pha);
```

```
    int c = cards.front();
    cards.pop_front();
    piles.push_back(piles.front());
    piles.pop_front();

    Pile& p = *(piles.back());
    p.push_back(c);
    procPile(p);
    if(p.empty()) piles.pop_back();

    return true;
}

int main()
{
    while(true) {
        cards.clear(), piles.clear(), phases.clear();
        _for(i, 0, CN) {
            int c = readint();
            if(c == 0) return 0;
            cards.push_back(c);
        }

        _for(i, 0, 7) {                //各个牌堆初始化
            Pile& p = allPiles[i];
            p.clear(), p.push_back(cards.front());
            cards.pop_front(), piles.push_back(&p);
        }

        int t = 7;
        while(true) if(!simulate(t++)) break;
    }
    return 0;
}
```

习题 6-11 树重建（Tree Reconstruction, UVa10410）

输入一个 n（$n\leqslant 1000$）结点树的 BFS 序列和 DFS 序列，你的任务是输出每个结点的儿子列表。输入序列（不管是 BFS 还是 DFS）是这样生成的：当一个结点被扩展时，它的所有孩子应该按照编号从小到大的顺序访问。

例如，若 BFS 序列为 4 3 5 1 2 8 7 6，DFS 序列为 4 3 1 7 2 6 5 8，则一棵满足条件的树如图 2.29 所示。

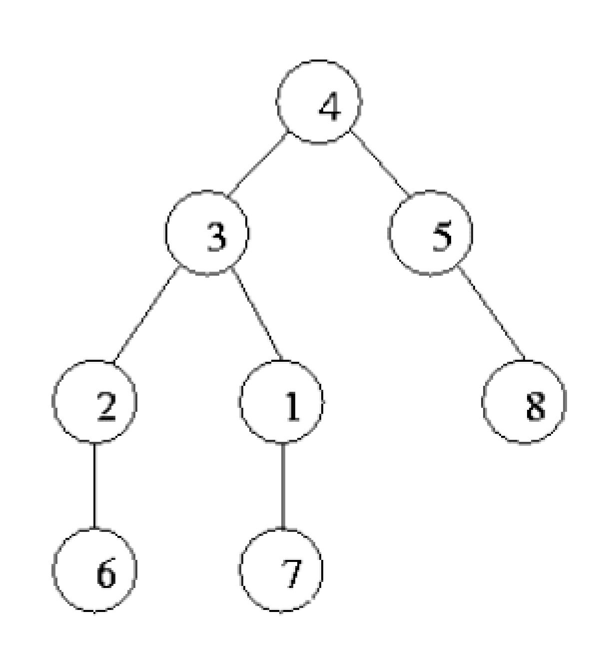


图 2.29

【分析】

根据题目描述可以得出结论：结点 u 及其直接孩子结点在 DFS 和 BFS 序列中出现的顺序一致且都是递增的，且孩子结点在 BFS 序列形成连续子序列。据此可以得到 u 的所有子结点，再根据子结点在 DFS 序列中的位置，就可以得到这两个子结点对应的子树对应的 DFS 序列以及子树的所有结点。

据此就可以设计出一个递归逻辑，读入一个子树的 DFS 序列，以及这个序列对应的 BFS 序列，构造这棵子树。

举例来说，对于以 4 为根结点的子树，输入的 BFS 和 DFS 序列分别是（灰色表示 4 的孩子结点）：BFS: 3 5 1 2 8 7，DFS: 3 1 7 2 6 5 8。则 4 的两个孩子结点分别是 3 和 5。根为 3 的子树对应的 BFS 和 DFS 序列就是：BFS: 1 2 7 6，DFS: 1 7 2 6。而 5 为根的子树对应的两个子序列就是 8。

根为 3 的子树，可以继续递归拆成两棵子树。

（1）子树 1 对应的子树序列：BFS: 7，DFS: 7。

（2）子树 2 对应的子树序列：BFS: 6，DFS: 6。

如此即可完整还原整棵树，算法的时间复杂度为 $O(n^2)$。完整程序（C++11）如下：

```
using namespace std;
int readint() { int x; cin >> x; return x; }
const int maxn = 1004;
int n, root[maxn];
vector<int> G[maxn], BFS, sub_dfs[maxn];

template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v){
    for(auto e : v) os<<" "<<e;
    return os;
}

void dfs(int u, int& bi){
   const auto& uDfs = sub_dfs[u];
   int sz = uDfs.size(), i = 0;
```

```
    while(i < sz) {
        int v = uDfs[i];
        if(bi < n && BFS[bi] == v) {              //v 是 u 的直接 child
            root[v] = u; G[u].push_back(v);
            bi++; i++;

            while(bi < n && i < sz) {
                int vv = uDfs[i];
                if(BFS[bi] == vv) break;
                sub_dfs[v].push_back(vv); root[vv] = v; i++;
            }
        }
    }

    while(bi < n) dfs(root[BFS[bi]], bi);
}

int main(){
    while (cin >> n && n) {
        BFS.clear();
        _rep(i, 0, n) G[i].clear(), sub_dfs[i].clear();
        _for(i, 0, n) BFS.push_back(readint());
        readint();
        _for(i, 1, n) sub_dfs[BFS[0]].push_back(readint());

        int bi = 1; dfs(BFS[0], bi);
        _rep(i, 1, n){
            sort(G[i].begin(), G[i].end());
            cout << i << ":" << G[i] << endl;
        }
    }
    return 0;
}
```

习题 6-12 骰子难题（A Dicey Problem, ACM/ICPC World Finals 1999, UVa810）

如图 2.30 所示是一个迷宫，如图 2.31 所示是一个骰子。你的任务是把骰子放在起点（骰子顶面和正面的数字由输入给定），经过若干次滚动以后回到起点。

每次到达一个新格子时，格子上的数字必须和它接触的骰子上的数字相同，除非到达的格子上画着五星（此时，与它接触的骰子上的数字可以任意）。输入一个 R 和 C 行（$1 \leqslant R,C \leqslant 10$）的迷宫、起点坐标以及顶面、正面的数字，输出一条可行的路径。

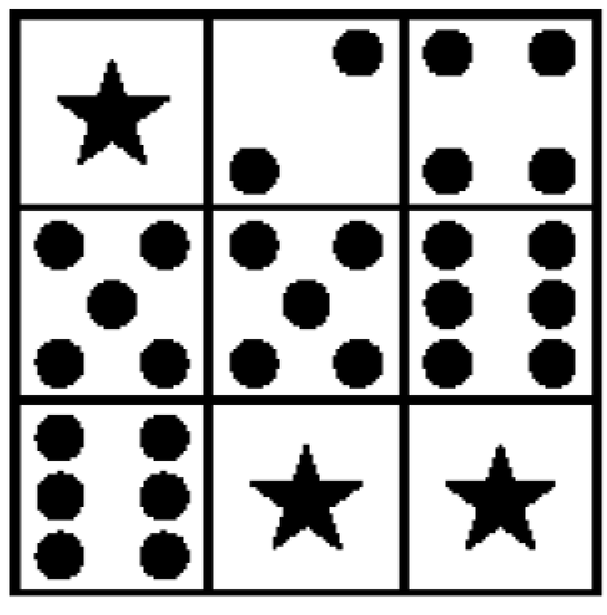

Figure 1: Sample Dice Maze

图 2.30

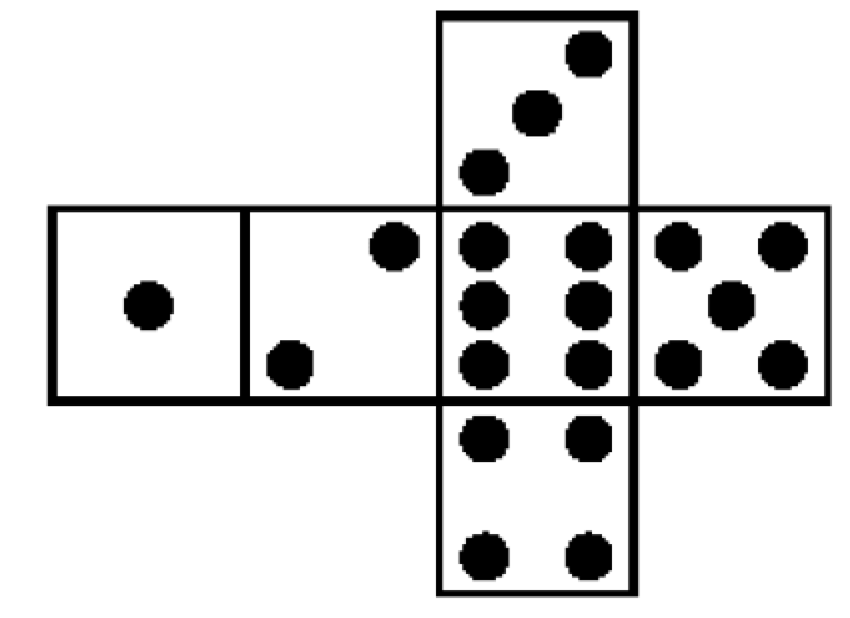

Figure 2: Standard Layout of Six-Sided Die

图 2.31

【分析】

首先要建立表示骰子当前状态的结构，其中包括当前的行列编号，以及顶面和正面的数字（由此可以确定其他 4 个面的数字）。然后就是使用 BFS 来寻找最短路径。

需要注意以下几点：

（1）在状态中还要保留指向上一步状态的指针。

（2）代码中手工打表来建立由顶面和正面数字得到左面数字的映射，这样初始状态输入时就直接建立 6 个面的状态。

（3）每次骰子旋转时，可以根据上一次的状态首先确认新的顶面和正面状态，然后即可确认 6 个面的状态。

完整程序如下：

```
using namespace std;
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
int readint() { int x; cin >> x; return x; }
enum DIR{ UP = 0, LEFT = 1, DOWN = 2, RIGHT = 3 };
const int MAXR = 12;
int R, C, M[MAXR][MAXR], dr[4] = { -1, 0, 1, 0 }, dc[4] = { 0, -1, 0, 1 };

template<typename T>
struct MemPool{/*此处省去*/};

//[face, top] -> left
int DL[6][6] = {
    { -1, 3, 5, 2, 4, -1 },        //1
    { 4, -1, 1, 6, -1, 3 },        //2
    { 2, 6, -1, -1, 1, 5 },        //3
    { 5, 1, -1, -1, 6, 2 },        //4
    { 3, -1, 6, 1, -1, 4 },        //5
    { -1, 4, 2, 5, 3, -1 }         //6
};
```



```
struct Stat {
    int r, c, face, top, back, bottom, left, right;
    Stat* prev;
    Stat() : prev(NULL) {}

    bool canMove(int dir);
    Stat* move(int dir);

    void init(int face, int top) {
        assert(face > 0 && face < 7);
        assert(top > 0 && top < 7);
        this->face = face;
        this->top = top;
        back = 7 - face;
        bottom = 7 - top;
        left = DL[face - 1][top - 1];
        assert(left > 0 && left < 7);
        right = 7 - left;
    }

    size_t hash() const {
        return 1000 * (r-1) + 100 * (c-1) + 10 * face + top;
    }
};
typedef Stat* PStat;
MemPool<Stat> pool;
struct PStatCmp {
    bool operator() (const PStat& lhs, const PStat& rhs) const {
        return lhs->hash() < rhs->hash();
    }
};

bool Stat::canMove(int dir) {
    assert(dir >= 0 && dir < 4);
    int nr = r + dr[dir], nc = c + dc[dir];
   if (nr < 1 || nr > R || nc < 1 || nc > C) return false;
   int m = M[nr][nc];
   if (m == 0) return false;
    return m == -1 || m == top;
}

PStat Stat::move(int d) {
```

```
    PStat ps = pool.createNew();
    ps->prev = this; ps->r = r + dr[d]; ps->c = c + dc[d];
    switch (d) {
    case UP:
        ps->init(bottom, face);
       assert(ps->left == left);
       assert(ps->right == right);
        break;
    case LEFT:
        ps->init(face, right);
       assert(ps->face == face);
        break;
    case DOWN:
        ps->init(top, back);
       assert(ps->left == left);
       assert(ps->right == right);
        break;
    case RIGHT:
        ps->init(face, left);
       assert(ps->face == face);
        break;
    default:
        assert(false);
    }

    return ps;
}

istream& operator>>(istream& is, Stat& s) {
    is >> s.r >> s.c >> s.top >> s.face;
    s.init(s.face, s.top);
    return is;
}

Stat* solve(const Stat& destS, PStat ps) {
    queue<PStat> q;
    set<PStat, PStatCmp> vis;
    q.push(ps); vis.insert(ps);
    while (!q.empty()) {
        PStat p = q.front(); q.pop();
        if (p->r == destS.r && p->c == destS.c)
            return p;
```

```
        _for(d, 0, 4) {
            if (!p->canMove(d)) continue;
            PStat np = p->move(d);
            if (vis.count(np)) continue;
            vis.insert(np);
            q.push(np);
        }
    }
    return NULL;
}

int main()
{
    string name;
   deque<PStat> outQ;
   char buf[64];

    while (cin >> name && name != "END") {
        Stat s;
        cin >> R >> C >> s;
        _for(r, 1, R + 1) _for(c, 1, C + 1) cin >> M[r][c];
        cout << name << endl;
        Stat* ans = NULL;
        _for(i, 0, 4) {
            if (s.canMove(i)) {
                ans = solve(s, s.move(i));
                if (ans) break;
            }
        }
        if (ans) {
          outQ.clear();
          while(ans) { outQ.push_front(ans); ans = ans->prev; }
          _for(i, 0, outQ.size()) {
              if(i) {
                  cout<<",";
                  if(i%9 == 0) cout<<endl;
              }

              if(i%9 == 0) cout<<"  ";
              cout<<"("<<outQ[i]->r<<","<<outQ[i]->c<<")";
          }
          cout<<endl;
```

```
        }
        else
            cout << "  No Solution Possible" << endl;
        pool.dispose();
    }
    return 0;
}
```

习题 6-13　电子表格计算器（Spreadsheet Calculator, ACM/ICPC World Finals 1992, UVa215）

在一个 R 行 C 列（$R\leqslant 20$，$C\leqslant 10$）的电子表格中，行编号为 A～T，列编号为 0～9。按照行优先顺序输入电子表格的各个单元格。每个单元格可能是整数（可能是负数）或者引用了其他单元格的表达式（只包含非负整数、单元格名称和加减号，没有括号）。表达式保证以单元格名称开头，内部不含空白字符，且最多包含 75 个字符。

尽量计算出所有表达式的值，然后输出各个单元格的值（计算结果保证为绝对值不超过 10000 的整数）。如果某些单元格循环引用，在表格之后输出（仍按行优先顺序），如图 2.32 所示。

样例输入	样例输出
2 2 A1+B1 5 3 B0-A1 3 2 A0 5 C1 7 A1+B1 B0+A1 0 0	0 1 A 3 5 B 3 -2 A0: A0 B0: C1 C1: B0+A1

图 2.32

【分析】

基本模型就是各个 Cell 之间的引用关系形成一个有向图，使用 DFS 递归求值，同时使用类似拓扑排序中的 DFS 逻辑来判断是否有循环引用。注意，表达式如果解析构造成树再递归计算，可能导致栈溢出。比较简洁的方法是直接对表达式进行解析，遇到 Cell 引用就递归计算对应的表达式同时判断循环引用，具体参见相关代码。另外，表达式可能以减号开头，解析时需要注意判断。完整程序如下：

```
using namespace std;
int readint() { int x; scanf("%d", &x); return x;}
```

```
const int MAXR = 20+1, MAXC = 10+1;

int R, C, RC, OK[MAXR][MAXC], Value[MAXR][MAXC];
char Exp[MAXR][MAXC][128];

int readint(const char* s, int& len) {
    len = 0;
    int base = 1, ans = 0;
    assert(isdigit(s[len]));
    while(s[len] && isdigit(s[len])) {
        ans *= base;
        ans += s[len] - '0';
        base *= 10;
        len++;
    }

    return ans;
}

bool eval(int r, int c) {
    int& o = OK[r][c];
    if(o == 1) return true;
    else if(o == -1) return false;

    o = -1;
    int& v = Value[r][c];
    v = 0;
    const char* s = Exp[r][c];
    int len = strlen(s), sign = 1;
    for(int i = 0; i < len; i++) {
        char ch = s[i];
        if(ch == '-') {  sign = -1;  }
        else if(ch == '+') { sign = 1; }
        else if(isdigit(ch)) {
            int len, value;
            value = readint(s+i, len);
            v += sign * value;
            sign = 1;
            i += len-1;
        }
        else if(isupper(ch)) {
            int len, col, row;
```

```
            col = readint(s+i+1, len);
            row = ch - 'A';
            if(!eval(row, col)) return false;
            v += sign * Value[row][col];
            sign = 1;
            i += len;
        }
    }
    o = 1;
    return true;
}

void solve() {
    memset(OK, 0, sizeof(OK));
    bool cycle = false;
    for(int i = 0; i < R; i++)
        for(int j = 0; j < C; j++)
        {
            bool o = eval(i,j);
            if(!o)
            {
                cycle = true;
                printf("%c%d: %s\n", 'A'+i, j, Exp[i][j]);
            }
        }

    if(cycle) return;
    printf(" ");
    for(int i = 0; i < C; i++)
        printf("%6d", i);
    puts("");

    for(int i = 0; i < R; i++)
    {
        printf("%c", i + 'A');
        for(int j = 0; j < C; j++)
            printf("%6d", Value[i][j]);
        puts("");
    }
}

int main()
```



```
{
    while(true) {
        R = readint(), C = readint(), RC = R*C;
        if(R == 0 || C == 0) break;
        for(int i = 0; i < R; i++)
            for(int j = 0; j < C; j++)
                scanf("%s", Exp[i][j]);

        solve();
        puts("");
    }
    return 0;
}
```

习题 6-14　检查员的难题（Inspector's Dilemma, ACM/ICPC Dhaka 2007, UVa12118）

某国家有 V（V≤1000）个城市，每两个城市之间都有一条双向道路直接相连，长度为 T。你的任务是找一条最短的道路（起点和终点任意），使得该道路经过 E 条指定的边。

例如，若 V=5、E=3、T=1，指定的 3 条边为 1-2、1-3 和 4-5，如图 2.33 所示，则最优道路为 3-1-2-4-5，长度为 4*1=4。

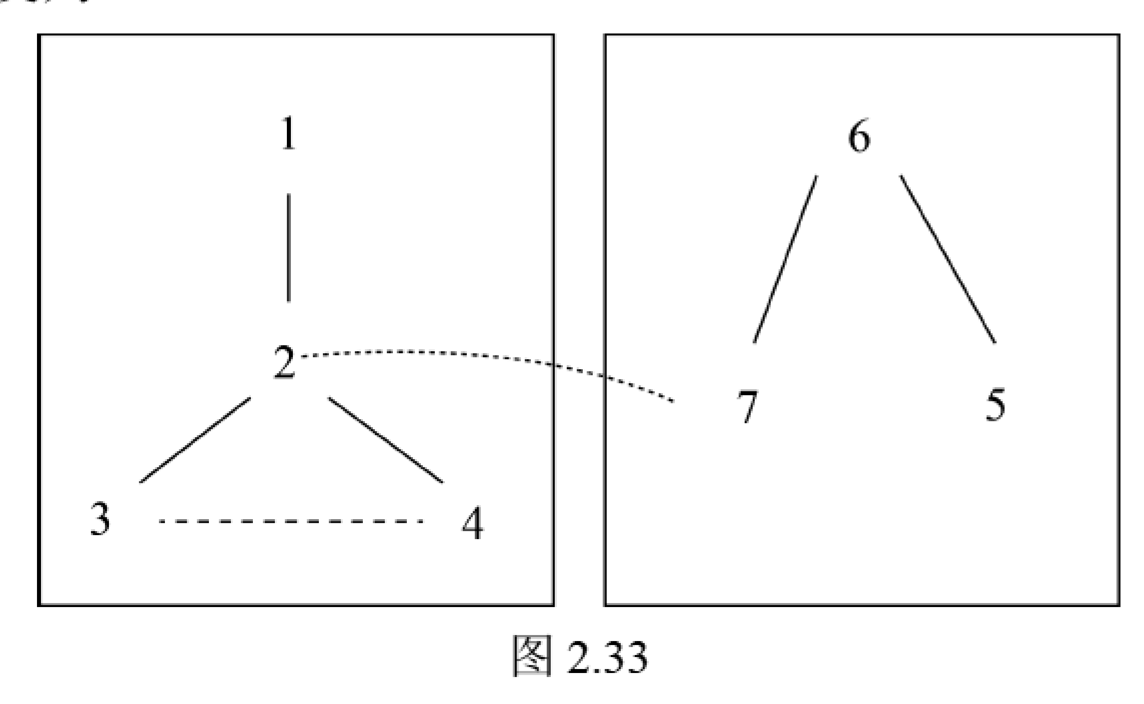


图 2.33

【分析】

首先考虑 E 条边都连通的情况。如果 E 条边组成的图存在欧拉道路（不需要是欧拉回路），则这条欧拉道路一定就是满足题目要求的最短道路。否则，记 G 中的奇度数点个数为 P，则一定有 P>2，最少需要使其中的 P−2 个点变成偶数度才能形成欧拉道路，而且题目强调了任意两个点都有一条边，那么可以增加(P−2)/2 条边来形成欧拉道路。这样欧拉道路的长度就是“T*(E 的边数+(P−2)/2)”。

需要强调的是，这里 P 一定是偶数，因为对于任意的无向图 G，所有点的度数之和等于所有边的端点个数之和。而每条边有两个端点，所有点的度数之和一定是偶数，那么奇度数点的度数之和也必然是偶数。把 P−2 个奇度数点两两连接起来就能保证存在欧拉道路。

如果 E 条边不连通，不妨设这 E 条边形成了 n 个连通分量 G，则需要首先要求每个 G 内部存在欧拉道路，不存在则参考单个连通分量的情况在 G 中构造欧拉道路。把每个 G 的欧拉道路首尾连接起来，至少需要 n−1 条边。所求答案就是 T*（n−1 + 每个 G 中的欧拉道

路长度之和）。而每个 G 要形成的欧拉道路长度和单个连通分量的情况类似。

举例来说，图 2.33 中存在两个连通分量：第一个中有 4 个奇度数点，不存在欧拉道路；第二个中无奇度数点，存在欧拉道路。则前者需要添加(4−2)/2=1 条边（即 3-4）形成欧拉道路。然后需要再添加一条边连接两个连通分量中的欧拉道路（即 2-7）。问题的解就是 7*T。完整程序如下：

```
using namespace std;
const int MAXV = 1004;
int V, E, T, Vis[MAXV];
vector<int> G[MAXV]; //E 条边组成的图

//dfs 遍历 u 所在的连通分量，返回这个点存在的奇度数点的个数
int dfs(int u) {
   if(Vis[u]) return 0; Vis[u] = 1;
   int sz = G[u].size(), r = sz%2;
   _for(i, 0, sz) r += dfs(G[u][i]);
   return r;
}

int main() {
   for(int a,b,t = 1; cin>>V>>E>>T && (V||E||T); t++) {
      _for(i, 0, V+1) G[i].clear();
      fill_n(Vis, V, 0);
      _for(i, 0, E) cin>>a>>b, G[a-1].push_back(b-1), G[b-1].push_back(a-1);

      int n = 0, resp = E; //连通分量个数，路径的长度
      _for(i, 0, V) {
         if(Vis[i] || G[i].empty()) continue;
         n++;
         resp += max(0, (dfs(i) - 2)/2);
      }
      printf("Case %d: %d\n", t, T*(resp + max(0, n-1)));
   }
   return 0;
}
```

2.5 暴力求解法

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 7 章。

习题 7-1　消防车（Firetruck, ACM/ICPC World Finals 1991, UVa208）

输入一个 n（$n \leqslant 20$）个结点的无向图以及某个结点 k，按照字典序从小到大顺序输出从结点 1 到结点 k 的所有路径，要求结点不能重复经过。

提示：

要实现判断结点 1 是否可以到达结点 k，否则会超时。

【分析】

如果 1 和 k 不连通，直接 DFS 搜索路径的话，一定会超时。而要判断 1 和 k 是否连通，使用《算法竞赛入门经典（第 2 版）》第 11.2.1 节中介绍的并查集即可。

本题建图时每个结点的邻居可以使用 set<int>存储，这样输入之后自然就是排好序的。使用 DFS 从 1 开始搜索到 k 的所有路径。因为结点不能重复经过，所以要在搜索过程中对已经经过的结点进行判重。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 20 + 4;
int N, pa[MAXN];
set<int> G[MAXN];

int find_pa(int x) { return pa[x] == x ? x : (pa[x] = find_pa(pa[x])); }

ostream& operator<<(ostream& os, const vector<int>& s){
   bool first = true;
   for(const auto x : s){
      if(first) first = false; else os<<' ';
      os<<x;
   }
   return os;
}
//搜索 src -> dest 的所有路径
void dfs(int src, int dest, vector<int>& path, vector<string>& paths) {
   path.push_back(src);
   if(src == dest) { //搜到目标了
      stringstream os; os<<path;
      paths.push_back(os.str());
   } else {
      for(auto v : G[src]) {   //遍历所有的下一步结点
         if(find(path.begin(), path.end(), v) != path.end())
            continue;          //走出的路径上已经有 v 存在了
         dfs(v, dest, path, paths);
      }
   }
```

```
        path.pop_back();
    }

    int main(){
        for(int kase = 1,from,to; scanf("%d", &N)==1; kase++){
            _for(i, 0, MAXN) G[i].clear(), pa[i] = i;
            while(true) {
                scanf("%d%d", &from, &to);
                if(from == 0 || to == 0) break;
                G[from].insert(to), G[to].insert(from);
                int pf = find_pa(from), pt = find_pa(to);
                if(pf != pt) pa[pt] = pf;
            }

            vector<string> paths;
            vector<int> path;
            if(find_pa(1) == find_pa(N)) dfs(1, N, path, paths);
            printf("CASE %d:\n", kase);
            for(auto& p : paths) puts(p.c_str());
            printf("There are %lu routes from the firestation to streetcorner %d.\n",
                paths.size(), N);
        }
        return 0;
    }
```

习题 7-2　黄金图形（Golygons, ACM/ICPC World Finals 1993, UVa225）

平面上有 k 个障碍点。从(0,0)点出发，第一次走 1 个单位，第二次走 2 个单位，……，第 n 次走 n 个单位，恰好回到(0,0)。要求只能沿着东南西北方向走，且每次必须转弯 90°（不能沿着同一个方向继续走，也不能后退）。走出的图形可以自交，但不能经过障碍点，如图 2.34 所示。

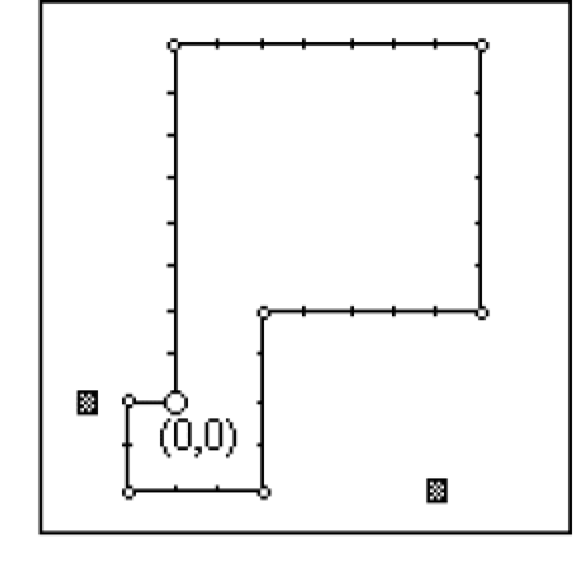


图 2.34

输入 n、k（$1 \leqslant n \leqslant 20$，$0 \leqslant k \leqslant 50$）和所有障碍点的坐标，输出所有满足要求的移动序列（用 news 表示北、东、西、南），按照字典序从小到大排列，最后输出移动序列的总数。

【分析】

还是典型的 DFS，不过需要注意以下几点：

（1）因为牵涉较多的坐标处理逻辑，使用二维几何中的 Point 类来表示位置以及向量。提前存储向 4 个方向走的 4 个向量，然后再回溯搜索。

（2）停留的点是不能重复的，这个需要在搜索时进行记录判重，因为最多走 20 步，所以坐标的最大值可能是 20*(20+1)/2 = 210，使用二维数组 Vis[512][512]进行判重，注意坐

标可能为负值，所以在判重之前先加上 256：vis[x+256][y+256]。

（3）剪枝优化：当前已经走了 k 步，共要走 n 步，则最多还能走出 $(n-k)$ * $(k+1+n)$ / 2 步，如果当前到(0,0)的距离大于这个数字，就肯定不能走到终点，直接返回即可。

完整程序（C++11）如下：

```
using namespace std;
//x in [left, right]
bool inRange(int x, int left, int right) {
   if(left > right) return inRange(x, right, left);
   return left <= x && x <= right;
}

ostream& operator<<(ostream& os, const vector<char>& s){
   for(const auto c : s) os<<c;
   return os;
}

vector<Point> blocks;                              //障碍点
char DIRS[] = "ensw";
Vector dirVs[4] = {{1,0},{0,1},{0,-1},{-1,0}};
unordered_map<char, int> DIX = {{'e', 0}, {'n', 1}, {'s', 2}, {'w', 3}};

string dbgPrintPath(const string& path) {
   Point pos;
   stringstream ss;
   _for(i, 0, path.size()) {
      char d = path[i];
      assert(DIX.count(d));
      ss<<'['<<pos.x<<','<<pos.y<<"]-"<<d<<"->";
      pos = pos + dirVs[DIX[d]]*(i+1);
   }
   ss<<endl;
   return ss.str();
}

//start--end 是否被 blocked
bool isBlocked(const Point& start, const Point& end) {
   assert(start.x == end.x || start.y == end.y);
   for(const auto& blk : blocks) {
      if(start.x == end.x) {
         if(start.x == blk.x && inRange(blk.y, start.y, end.y))
```

```
                return true;
            }
            else if(start.y == end.y) {
                if(start.y == blk.y && inRange(blk.x, start.x, end.x))
                    return true;
            }
        }
        return false;
    }
    const int MAXX = 256;
    int vis[MAXX*2][MAXX*2];
    void solve(const Point& pos, vector<char>& path, vector<string>& paths, int
cities) {
        int n = path.size();
        if(n == cities) {
            if(pos.x == 0 && pos.y == 0) {                    //已经回家了
                stringstream ss; ss<<path;
                paths.push_back(ss.str());                    //搜到一条路径
            }
            return;
        }

        int dist = abs(pos.x) + abs(pos.y),                   //距离起点的距离
            walks = (cities - n) * (n + 1 + cities) / 2;      //还能走出的步数
        if(walks < dist) return;                              //怎么走都走不到终点

        _for(i, 0, 4) {
            char d = DIRS[i];                                 //下一步走什么方向
            if(n) {
                char lastD = path.back();
                if(lastD == d) continue;
                if(lastD == 'e' && d == 'w') continue;        //反向不行
                if(d == 'e' && lastD == 'w') continue;        //反向不行
                if(lastD == 'n' && d == 's') continue;        //反向不行
                if(d == 'n' && lastD == 's') continue;        //反向不行
            }
            auto dest = pos + dirVs[i]*(n+1);                 //这一步的目标点
            if(isBlocked(pos, dest)) continue;
            int& destVis = vis[dest.x+MAXX][dest.y+MAXX];
            if(destVis) continue;                             //已经在目标点停留过
            destVis = 1;
```



```
        path.push_back(d);
        solve(dest, path, paths, cities);
        path.pop_back();
        destVis = 0;
    }
}

int main(){
    int N,k,K; scanf("%d", &K);
    Point b;
    while(K--){
        blocks.clear();
        scanf("%d%d", &N, &k);
        _for(i, 0, k){
            scanf("%d%d", &(b.x), &(b.y));
            blocks.push_back(b);
        }

        Point start; vector<string> paths; vector<char> path;
        memset(vis, 0, sizeof(vis));
        solve(start, path, paths, N);
        for(const auto& p : paths) puts(p.c_str());
        printf("Found %lu golygon(s).\n\n", paths.size());
    }
    return 0;
}
```

习题 7-3　多米诺效应（The Domino Effect, ACM/ICPC World Finals 1991, UVa211）

一副"双六"多米诺骨牌包含 28 张，编号如图 2.35 所示。

Bone #	Pips	Bone #	Pips	Bone #	Pips	Bone #	Pips
1	0 | 0	8	1 | 1	15	2 | 3	22	3 | 6
2	0 | 1	9	1 | 2	16	2 | 4	23	4 | 4
3	0 | 2	10	1 | 3	17	2 | 5	24	4 | 5
4	0 | 3	11	1 | 4	18	2 | 6	25	4 | 6
5	0 | 4	12	1 | 5	19	3 | 3	26	5 | 5
6	0 | 5	13	1 | 6	20	3 | 4	27	5 | 6
7	0 | 6	14	2 | 2	21	3 | 5	28	6 | 6

图 2.35

在 7*8 网格中每张牌各摆一张，如图 2.36 所示，左边是各个格子的点数，右边是各个格子所属的骨牌编号。

```
7 x 8 grid of pips              map of bone numbers

6  6  2  6  5  2  4  1          28 28 14  7 17 17 11 11
1  3  2  0  1  0  3  4          10 10 14  7  2  2 21 23
1  3  2  4  6  6  5  4           8  4 16 25 25 13 21 23
1  0  4  3  2  1  1  2           8  4 16 15 15 13  9  9
5  1  3  6  0  4  5  5          12 12 22 22  5  5 26 26
5  5  4  0  2  6  0  3          27 24 24  3  3 18  1 19
6  0  5  3  4  2  0  3          27  6  6 20 20 18  1 19
```

图 2.36

输入图 2.36 所示左图，你的任务是输出所有可能的如图 2.36 右图所示的结果。

【分析】

使用回溯法，对网格中的坐标从左到右、从上到下进行遍历，每一步考虑水平和垂直放置两种方法，如果这个格子已经被之前的骨牌占用，则直接遍历到下一个格子。

注意以下几点：

（1）对骨牌的面值进行索引，可以根据牌面的两个数值查找对应的骨牌编号。

（2）对“按顺序跳到下一个 Cell”的逻辑进行封装。

（3）在回溯过程中要记录已经放置过的骨牌编号。

```
using namespace std;

int readint() { int x; scanf("%d", &x); return x;}
//x in [left, right]
bool inRange(int x, int left, int right) {
    if(left > right) return inRange(x, right, left);
    return left <= x && x <= right;
}
struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
};
typedef Point Bone;

const int ROW = 7, COL = 8, BoneCnt = 28;
int Grid[ROW][COL], Result[ROW][COL];
int boneIndice[ROW][ROW];                //牌面为(x,y)的骨牌编号
Bone bones[BoneCnt];

//跳到下一个格子，到行尾换行，走到最后一格就返回 false
bool gotoNextCell(Point& pos){
    int r = pos.x, c = pos.y;
    assert(inRange(r, 0, ROW-1));
    assert(inRange(c, 0, COL-1));
```

```
    c++;
    r += c / COL;
    c %= COL;
    if(r >= ROW) return false;
    pos.x = r; pos.y = c;
    return true;
}

void initBones() {
    int cur = 0;
    memset(boneIndice, -1, sizeof(boneIndice));
    _for(i, 0, ROW) {
        _for(j, i, ROW) {
            Bone& b = bones[cur];
            b.x = i; b.y = j;
            boneIndice[i][j] = cur++;
        }
    }
}

int findBone(int p1, int p2) {
    if(p1 > p2) return findBone(p2, p1);
    return boneIndice[p1][p2];
}

void solve(const Point& pos, set<int> usedBones, int& ansCnt) {
    if(usedBones.size() == BoneCnt) {
        ansCnt++;
        _for(i, 0, ROW) {
            printf(" ");
            _for(j, 0, COL) printf("%4d", Result[i][j]+1);
            printf("\n");
        }
        printf("\n");
        return;
    }

    int r = pos.x, c = pos.y;
    assert(inRange(r, 0, ROW-1)); assert(inRange(c, 0, COL-1));

    Point np = pos;
    if(Result[r][c] != -1) {                    //这个格子已经被决策了
```

```
        if(gotoNextCell(np)) solve(np, usedBones, ansCnt);
        return;
    }

    np = pos;
    if(!gotoNextCell(np)) return;           //决策下一个格子

    //水平放骨牌
    if(c+1 < COL && Result[r][c+1] == -1) {
        int b = findBone(Grid[r][c], Grid[r][c+1]);
        if(b != -1 && !usedBones.count(b)) {
            np = pos;
            usedBones.insert(b);
            Result[r][c] = Result[r][c+1] = b;
            assert(gotoNextCell(np));

            solve(np, usedBones, ansCnt);

            usedBones.erase(b);
            Result[r][c] = Result[r][c+1] = -1;
        }
    }

    //垂直放骨牌
    if(r+1 < ROW) {
        assert(Result[r+1][c] == -1);
        int b = findBone(Grid[r][c], Grid[r+1][c]);
        if(b != -1 && !usedBones.count(b)) {
            np = pos;
            usedBones.insert(b);
            Result[r][c] = Result[r+1][c] = b;
            assert(gotoNextCell(np));

            solve(np, usedBones, ansCnt);

            usedBones.erase(b);
            Result[r][c] = Result[r+1][c] = -1;
        }
    }
}

int main()
```



```
{
    initBones();
    int t = 1;
    while(true) {
        Point pos;
        if(scanf("%d", &(Grid[pos.x][pos.y])) != 1) break;
        while(gotoNextCell(pos)) Grid[pos.x][pos.y] = readint();

        if(t > 1) printf("\n\n\n");
        printf("Layout #%d:\n\n", t);
        _for(i, 0, ROW){
            _for(j, 0, COL) printf("%4d", Grid[i][j]);
            printf("\n");
        }
        printf("\nMaps resulting from layout #%d are:\n\n", t);

        int ansCnt = 0;
        memset(Result, -1, sizeof(Result));
        set<int> usedBones;
        solve(Point(), usedBones, ansCnt);
        printf("There are %d solution(s) for layout #%d.\n", ansCnt, t++);
    }
    return 0;
}
```

习题 7-4　切断圆环链（Cutting Chains, ACM/ICPC World Finals 2000, UVa818）

有 n（$n \leqslant 15$）个圆环，其中有一些已经扣在了一起。现在需要打开尽量少的圆环，使得所有圆环可以组成一条链（当然，所有打开的圆环最后都要再次闭合）。例如有 5 个圆环，如 1-2, 2-3, 4-5，则需要打开一个圆环，如圆环 4，然后用它穿过圆环 3 和圆环 5 后再次闭合圆环 4，就可以形成一条链：1-2-3-4-5。

【分析】

关键点是把所有圆环打开之后，肯定能形成一条链，所以问题一定有解。因为 $n \leqslant 15$，可以用位向量表示并遍历每个圆环是否打开，记 kc 为打开的圆环的个数。确定这个集合之后，可把每个未打开的圆环看作一个结点，如果两个圆环是相连的就在图中形成一条无向边，则结点和边组成的图（可能有多个点连通分量）符合以下 3 个条件即可形成一条链。

（1）不能有分叉，也就是说结点的度数都小于等于 2。

（2）不能有环，可使用 DFS 判圈，因为图为无向图，所以在 DFS 到每个点时要传入父结点。

（3）每个打开的环只能连接两个未打开的环，也就是两个连通分量。所以当前连通分量的个数 ti 必须满足 $ti \leqslant kc + 1$。ti 可以在判圈时一并求出。

本题中使用 STL 中的 bitset 来作为位向量。注意在判圈和求度数时，必须忽略已经打开的环。完整程序如下：

```
int G[MAXN][MAXN], n, C[MAXN];

//结点 i 的度数(i 是对应未打开的圆环)
int degree(int i, const bitset<MAXN>& opened) {
    int ans = 0;
    if (opened.test(i)) return ans;
    _for (j, 0, n) if (!opened.test(j) && G[i][j]) ans++;
    return ans;
}

//DFS 判圈，返回结点 i 是否存在圈
bool dfs(const int i, const int pa, const bitset<MAXN>& opened) {
    if (C[i] == 1) return true;                              //已经判断完成
    if (C[i] == -1) return false;                            //正在判圈
    if (opened.test(i)) { C[i] = 1; return true; }           //不考虑打开的圆环
    C[i] = -1;
    _for (j, 0, n) if (G[i][j] && j != pa && !dfs(j, i, opened)) return false;
    C[i] = 1;
    return true;
}

int solve() {
    int ans = n;
    bitset<MAXN> opened;
    _for (k, 0, 1<<n) {
        opened.reset();
        int valid = true;
        _for (i, 0, n) if (k & (1 << i)) opened.set(i);

        _for (i, 0, n)
            if (!opened.test(i) && degree(i, opened) > 2) { valid = false;
break; }
        if (!valid) continue;

        fill_n(C, MAXN, 0);
        int ti = 0;                                          //连通分量的个数
        _for (i, 0, n) {
            if (opened.test(i)) continue;
            if (!C[i]) ti++;
            if (!dfs(i, -1, opened)) { valid = false; break; }
        }
       if(!valid) continue;
```

```
            int kc = opened.count(); // cout<<"k = "<<k<<", kc = "<<kc<<" no loop, ti = "<<ti<<endl;
            if (ti <= kc + 1) ans = min(ans, kc);
        }
        return ans;
    }

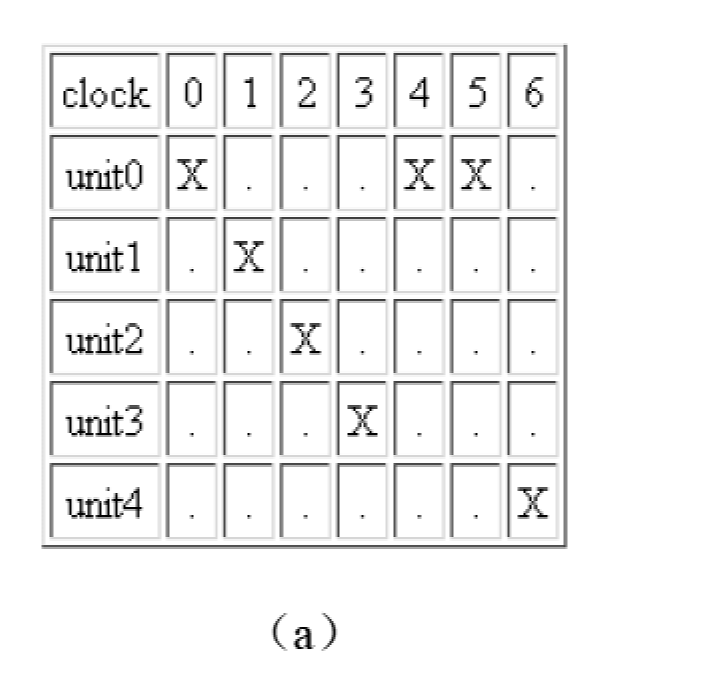
    int main(){
        for (int t = 1; scanf("%d", &n) == 1 && n; t++) {
            memset(G, 0, sizeof(G));
            int from, to;
            while (true) {
                scanf("%d%d", &from, &to);
                if (from == -1 || to == -1) break;
                from--; to--;
                G[from][to] = G[to][from] = 1;
            }
            int ans = solve();
            printf("Set %d: Minimum links to open is %d\n", t, ans);
        }

        return 0;
    }
```

习题 7-5　流水线调度（Pipeline Scheduling, UVa690）

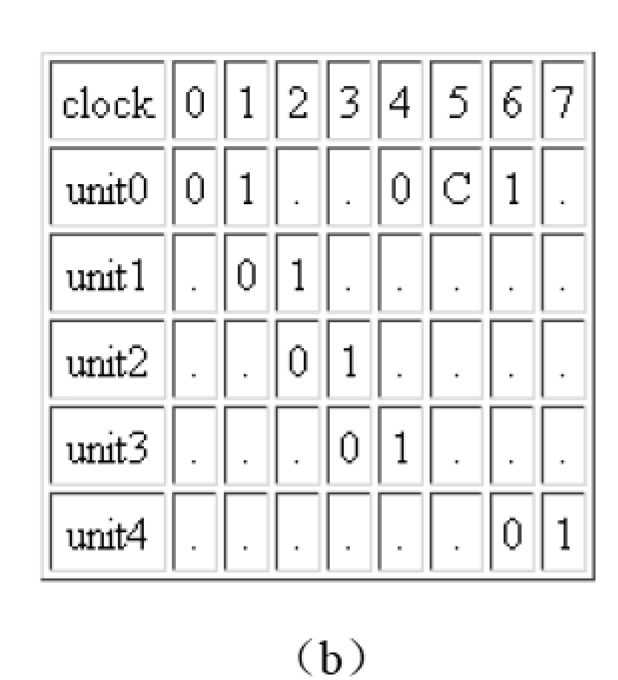
给 10 个完全相同的任务安排一个流水线调度。输入数据是如图 2.37（a）所示的 reservation table。在图 2.37 中，在时间 4 时 unit0 处于工作状态。

在你的流水线调度中不能同时有两个任务使用同一个 unit。例如若两个任务分别在时间 0 和 1 开始，则在时间 5 时 unit0 会发生冲突，如图 2.37（b）所示。



clock	0	1	2	3	4	5	6
unit0	X	.	.	.	X	X	.
unit1	.	X	.	.	.	.	.
unit2	.	.	X	.	.	.	.
unit3	.	.	.	X	.	.	.
unit4	.	.	.	.	.	.	X

（a）



clock	0	1	2	3	4	5	6	7
unit0	0	1	.	.	0	C	1	.
unit1	.	0	1	.	.	.	.	.
unit2	.	.	0	1	.	.	.	.
unit3	.	.	.	0	1	.	.	.
unit4	.	.	.	.	.	.	0	1

（b）

图 2.37

输入一个 5 行 n（n<20）列的 resrevation table，输出 10 个任务执行完毕所需的最少时间。如上面的例子，答案为 34。

【分析】

初步看，就是一个回溯，依次对每个任务的开始时间进行决策，判断有无冲突。这样算法复杂度为 $10 * 10^{10}$，肯定超时。但是注意本题中的所有任务都完全相同，在这个前提下，考虑使用以下剪枝：

（1）两个任务的开始时间可以有不同的间隔（1～n），但是有的间隔会引起冲突。提前计算出两个任务之间所有的合法间隔，进行决策时只是用合法间隔进行下一个任务的安排。

（2）此类搜索问题中，常见的一个剪枝技巧就是已经决策了 i 个，已用的时间为 t，判断 t 加上剩下的 10−i 所需的时间是否已经超过当前搜索出的最优时间，如果超过，则直接返回。而所有任务完全相同，因此可以想到将“i 个任务调度好所需要的最小时间”记录为 A_i。一开始令所有 A_i=i*n，然后从 i = 1～10 依次求 A_i。而求 A_i 的过程中就可以复用 A_{10-i} 来进行上述剪枝。

最终 A_{10} 就是问题答案。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 20+1, UNITS = 5, MAXT = 10;
int n, S[UNITS][MAXT*MAXN];
int Task[MAXN];                         //Task[i]第 i 个时间点是在 unit
int Ans[MAXT+1];
vector<int> Dist;

//能不能在 clock 开始执行一个任务
bool canPut(int clock) {
   _for(i, 0, n) if(S[Task[i]][clock+i]) return false;
   return true;
}
//在 clock 开始安排一个任务
void put(int clock) { _for(i, 0, n) S[Task[i]][clock+i] = 1; }
//清除从 clock 开始安排的任务
void remove(int clock){ _for(i, 0, n) S[Task[i]][clock+i] = 0; }
//已经安排了 t 个任务，共安排 T 个任务，第 t+1 个任务从 clock 开始执行
void dfs(int t, int T, int clock, int& ans) {
   if(t == T) {
      ans = min(ans, clock + n);
      return;
   }

   for(const auto D : Dist) {
      int c = clock + D;
      if(c + Ans[T-t] >= ans) break;   //无论如何不会搜出最优答案了
      if(!canPut(c)) continue;
```



```
        put(c);
        dfs(t+1, T, c, ans);
        remove(c);
    }
}

int main()
{
    char line[64];
    while(scanf("%d", &n) == 1 && n){
        memset(S, 0, sizeof(S)), memset(Task, 0, sizeof(Task)), memset(Ans,
0, sizeof(Ans));
        Dist.clear();
        _for(i, 0, UNITS){
            scanf("%s", line);
            _for(j, 0, n){
                if(line[j] == 'X'){
                    assert(!Task[j]);
                    Task[j] = i;
                }
            }
        }

        put(0);
        _for(i, 1, n+1) if(canPut(i)) Dist.push_back(i); //两个任务可以间隔 i

        Ans[1] = n;
        _for(T, 1, MAXT+1) {
            Ans[T] = T * n;                         //T 个任务执行完需要的最优时间
            dfs(1, T, 0, Ans[T]);
        }

        printf("%d\n", Ans[MAXT]);
    }
    return 0;
}
```

习题 7-6 重叠的正方形（Overlapping Squares, Xi'an 2006, UVa12113）

给出一个 4*4 的棋盘和棋盘上所呈现出来的纸张边缘，问用不超过 6 张 2*2 的纸能不能摆出这样的形状，如图 2.38 所示。

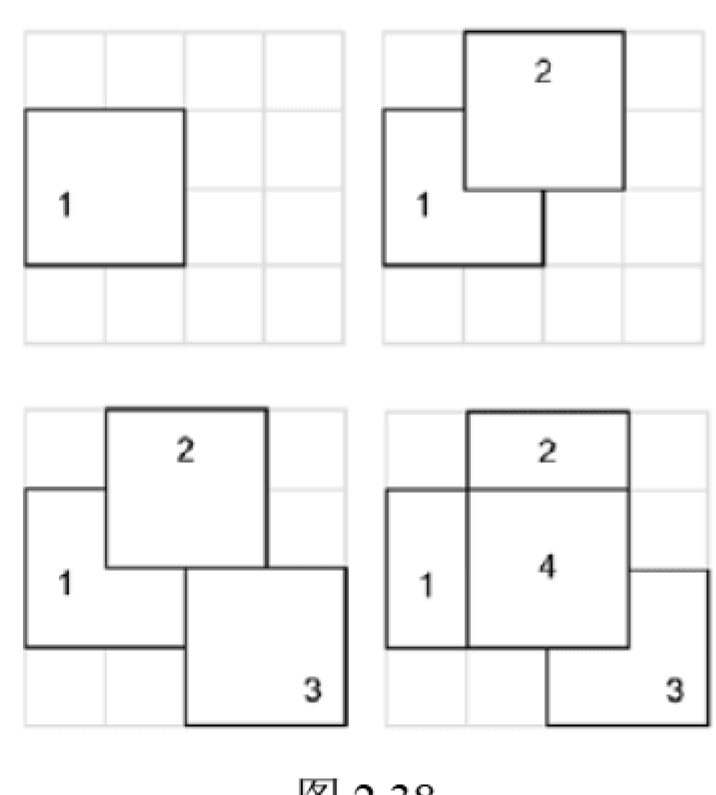


图 2.38

【分析】

关键在于棋盘的建模，因为题目需要考虑正方形的边以及正方形之间的覆盖。可以将棋盘抽象成一个 5*5 的 Grid，Grid 中每个点有两个状态值：从这个点开始向下延伸的边以及向右延伸的边分别是否可见。这样 Grid 的状态个数就是 5*5*2=50 个，对应的，分别使用两个 32 位整数作为二进制集合即可。

读取输入之后建立目标局面 target。然后进行回溯，从左到右，从上到下，每次尝试在所有可能的位置放置一张纸，看看能不能形成目标局面，最多放 6 张纸。完整程序如下：

```
using namespace std;
struct Grid {
    int HEdges, VEdges;
    inline void clear() { HEdges = 0, VEdges = 0; }
    inline bool getHEdge(int row, int col) const { return HEdges & (1<<(row*5+col)); }
    inline bool getVEdge(int row, int col) const { return VEdges & (1<<(row*5+col)); }

    inline void setHEdge(int row, int col) { HEdges |= (1<<(row*5 + col)); }
    inline void clearHEdge(int row, int col) { HEdges &= ~(1<<(row*5 + col)); }
    inline void setVEdge(int row, int col) { VEdges |= (1<<(row*5 + col)); }
    inline void clearVEdge(int row, int col) { VEdges &= ~(1<<(row*5 + col)); }

    Grid() { clear(); }
    inline bool operator==(const Grid& g) const {
        return HEdges == g.HEdges && VEdges == g.VEdges;
    }

    void putSquare(int r, int c) { //以 r,c 为左上角放一张纸
        assert(0 <= r && r <= 2);
        assert(0 <= c && c <= 2);
```



```
        setHEdge(r,c), setVEdge(r,c), setHEdge(r,c+1);
        clearVEdge(r,c+1);
        setVEdge(r,c+2), setVEdge(r+1,c);
        clearHEdge(r+1,c), clearHEdge(r+1,c+1), clearVEdge(r+1,c+1);
        setVEdge(r+1,c+2), setHEdge(r+2,c), setHEdge(r+2,c+1);
    }
};

ostream& operator<<(ostream& os, const Grid& g) {
    _for(r, 0, 5) {
        _for(c, 0, 5) {
            os<<((r && g.getVEdge(r-1, c)) ? '|' : ' ');
            os<<(g.getHEdge(r,c)?'_':' ');
        }
        os<<"#"<<endl;
    }
    return os;
}

Grid target;
//g：目前已经放好的 Grid 布局；dep：已经放上去的纸张个数
bool dfs(const Grid& g, int dep) {
    if(g == target) return true;
    if(dep >= 6) return false;
    _for(r, 0, 3) _for(c, 0, 3){
        Grid ng = g;
        ng.putSquare(r, c);                 //新的局面
        if(dfs(ng, dep + 1)) return true;
    }
    return false;
}

int main(){
    string line;
    for(int k = 1; ;k++){
        target.clear();
        _for(i, 0, 5){
            getline(cin, line);
            if(line == "0") return 0;
            _for(j, 0, 9){
                switch(line[j]){
```

```
                    case ' ':
                        break;
                    case '_':
                        assert(j%2);
                        target.setHEdge(i, j/2);
                        break;
                    case '|':
                        assert(j%2==0);
                        target.setVEdge(i-1, j/2);
                        break;
                    default:
                        cout<<"c = "<<line[j]<<endl;
                        assert(false);
                }
            }
        }
        Grid g;
        bool ans = dfs(g, 0);
        cout<<"Case "<<k<<": "<<(ans?"Yes":"No")<<endl;
    }
    return 0;
}
```

习题 7-10 守卫棋盘（Guarding the Chessboard, UVa11214）

输入一个 $n*m$ 棋盘（$n,m<10$），某些格子有标记。用最少的皇后守卫（即占据或者攻击）所有带标记的格子。

【分析】

这个题目也可以像八皇后问题一样使用回溯法搜索，不过关键也是棋盘的编码以及决策的顺序。

（1）棋盘最大有 9*9 个格子，也就是用 3 个 32bit 的 int 足以表示每一个格子的状态（被覆盖与否）。本题的时间限制比较紧，状态值如果用数组存储，状态转移以及最终判断棋盘覆盖是否为可行解就需要做循环赋值或者判等操作，会导致 TLE。

（2）在决策时，按照从左到右，从上到下的顺序依次每个格子进行决策：是否放皇后。

（3）注意每个位置放皇后时因为要设置一系列的 bit 值（行列），可以把每个位置放皇后要设置的所有 bit 值也预先存到 3 个 int 内，然后进行一次“或”的位运算，即可完成状态转移。否则还要循环对每个位赋值，也会导致超时，这也是本题需要使用 3 个 int 作为位集合表示棋盘状态的最重要原因。

完整程序如下：

```
using namespace std;
```

```
int n, m;
struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
};
typedef Point Vector;
const int MAXM = 9;

Vector operator+ (const Vector& A, const Vector& B) { return Vector(A.x+B.x,
A.y+B.y); }
Vector dirs[] = //8 个方向向量
    {{-1, -1},{1, 1}, {1, -1},{-1, 1}, {1, 0},{-1, 0}, {0, 1},{0, -1}};
bool isValid(const Point& p) { return p.x >= 0 && p.x < n && p.y >= 0 && p.y
< m; }

struct Grid{
    int bits[4];
    inline void clear() { memset(bits, 0, sizeof(bits)); }

    inline void set(int r, int c) {
        int l = r*m+c;
        bits[l/32] |= (1<<(l&31));
    }

    inline bool canCover(const Grid& g) const {
        return (bits[0]&g.bits[0]) == g.bits[0]
            && (bits[1]&g.bits[1]) == g.bits[1]
            && (bits[2]&g.bits[2]) == g.bits[2];
    }

    Grid() { clear(); }
};

//[i,j]如果有皇后，所有被其覆盖的布局就是 covers[i*m+j]
Grid covers[MAXM*MAXM + 3], target;

void dfs(int ci, const Grid& g, int depth, int& best) {
    if(depth >= best) return;
    if(g.canCover(target)) {
        best = min(best, depth);
        return;
    }
```

```
    if(ci > n*m || depth + 1 > best) return;

    dfs(ci+1, g, depth, best);
    Grid ng = g;
    int *cb = covers[ci].bits;
    ng.bits[0] |= cb[0], ng.bits[1] |= cb[1], ng.bits[2] |= cb[2];
    dfs(ci+1, ng, depth+1, best);
}

int main(){
    string line;
    for(int k = 1; cin>>n>>m && n && m; k++) {
        target.clear();
        memset(covers, 0, sizeof(covers));
        _for(i, 0, n) {
            cin>>line;
            assert(line.size() == m);
            _for(j, 0, m){
                if(line[j] == 'X') target.set(i, j);
                for(const auto& dv : dirs) {
                    Point pc(i,j);
                    while(isValid(pc)) {
                        covers[i*m+j].set(pc.x, pc.y);
                        pc = pc + dv;
                    }
                }
            }
        }

        int best = 6;
        Grid g;
        dfs(0, g, 0, best);
        cout<<"Case "<<k<<": "<<best<<endl;
    }
    return 0;
}
```

习题 7-11　树上的机器人规划（简单版）（Planning mobile robot on Tree (EASY Version), UVa12569）

有一棵 n（$4 \leqslant n \leqslant 15$）个结点的树，其中一个结点有一个机器人，还有一些结点有石头。每步可以把一个机器人或者石头移到一个相邻结点。任何情况下一个结点里不能有两个东西（石头或者机器人）。输入每个石头的位置和机器人的起点和终点，求最小步数的方案。



如果有多解，可以输出任意解。如图 2.39 所示，s=1，t=5，最少需要 16 步：机器人 1-6，石头 2-1-7，机器人 6-1-2-8，石头 3-2-1-6，石头 4-3-2-1，最后机器人 8-2-3-4-5。

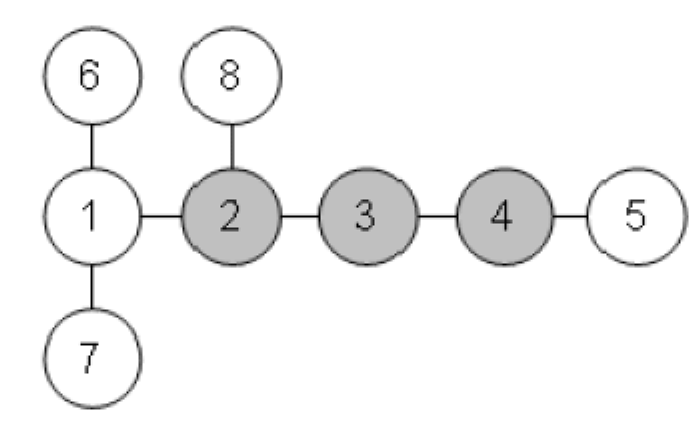


图 2.39

【分析】

看到最小步数问题，首先想到肯定是用 BFS。其次是状态编码表示，使用一个 int，最低 4 位就可以表示机器人的位置。然后剩下的位表示每个结点上是否有石头，这样就可以用一个 int 数组来进行状态判重。可将设置状态各个部分的位运算代码封装起来。每次考虑是机器人移动还是石头移动，尝试往各个方向移动，生成新的状态进行搜索即可。

```
using namespace std;
int readint() { int x; cin>>x; return x; }
template<typename T>
struct MemPool {
   vector<T*> buf;
   T* createNew() {
      buf.push_back(new T());
      return buf.back();
   }

   void dispose() {
      for(int i = 0; i < buf.size(); i++) delete buf[i];
      buf.clear();
   }
};
const int MAXN = 16;

struct Node {                          //表示路径的链表结点
   int from, to;
   Node* next;
};

struct State {
   Node* path;                         //路径
   int g, len;                         //状态压缩，路径长度
   State(int gi = 0, int li = 0, Node* pn = NULL) : g(gi), len(li), path(pn) {}
   inline bool operator[](size_t i) const { return g&(1<<(i+4)); }
                                       //位置 i 上是否有石头
   inline void setRock(size_t i, bool val = true) { //设置位置 i 上是否有石头
      if(val) g |= 1<<(i+4);
```

```
        else g &= ~(1<<(i+4));
    }
    //机器人的位置操作
    inline int getP() const { return g&15; }
    inline void setP(int p) { g = ((g>>4)<<4)|p; }
};

vector<int> G[MAXN];                                        //图的邻接矩阵表示
MemPool<Node> pool;                                         //链表结点分配
int n,m,S,T, O[MAXN], VIS[1<<19];
Node* newNode(Node* next = NULL, int u = -1, int v = -1) {
    Node* p = pool.createNew();
    p->next = next, p->from = u, p->to = v;
    return p;
}

ostream& operator<<(ostream& os, Node* p) {
    if(p == NULL) return os;
    os<<p->next<<p->from+1<<" "<<p->to+1<<endl;
    return os;
}

//尝试移动在点 from 上的物体（机器人或者石头）
void tryMove(const State& s, int from, queue<State>& q) {
    int rp = s.getP();
    assert(from >= 0 && from < n);
    for(auto to : G[from]) {
        if(to == rp || s[to]) continue;              //目标点有石头或机器人
        int ng = s.g;
        if(from == rp) ng = ((s.g>>4)<<4)|to;        //移动机器人
        else ng ^= (1<<(from+4)), ng ^= (1<<(to+4));//移动石头
        if(VIS[ng]) continue;                        //新的状态已经访问过
        VIS[ng] = 1;
        q.push(State(ng, s.len+1, newNode(s.path, from, to)));
    }
}

void solve() {
    State s;
    _for(i, 0, m) s.setRock(O[i]);
    s.setP(S);
    queue<State> q;
```

```
    q.push(s);
    VIS[s.g] = 1;
    while(!q.empty()) {
        const State& st = q.front();
        int rp = st.getP();
        if(rp == T) {                                    //到达目的地
            cout<<st.len<<endl<<st.path;
            return;
        }
        tryMove(st, rp, q);                              //尝试移动机器人
        _for(i, 0, n) if(st[i]) tryMove(st, i, q);  //尝试移动石头
        q.pop();
    }
    cout<<"-1"<<endl;
}

int main()
{
    int K = readint();
    for(int t = 1; t <= K; t++) {
        memset(VIS, 0, sizeof(VIS));
        cin>>n>>m>>S>>T;
        --S; --T;
        cout<<"Case "<<t<<": ";
        _for(i,0,m) O[i] = readint()-1;
        _for(i,0,n) G[i].clear();

        _for(i, 0, n-1) {
            int u = readint()-1, v = readint()-1;
            G[u].push_back(v); G[v].push_back(u);
        }

        solve();
        pool.dispose();
        cout<<endl;
    }
    return 0;
}
```

习题 7-12 移动小球（Moving Pegs, ACM/ICPC Taejon 2000, UVa1533）

如图 2.40 所示，一共有 15 个洞，其中一个空着，剩下的洞里各有一个小球。每次可以让一个小球越过同一条直线上的一个或多个小球后跳到最近的空洞中，然后拿走被跳过的

小球。如让 14 跳到空洞 5 中，则洞 9 里的小球会被拿走，因此操作之后洞 9 和 14 会变空，而 5 里面会有一个小球。你的任务是用最少的步数让整个棋盘只剩下一个小球，并且这个小球留在最初输入的空洞中。

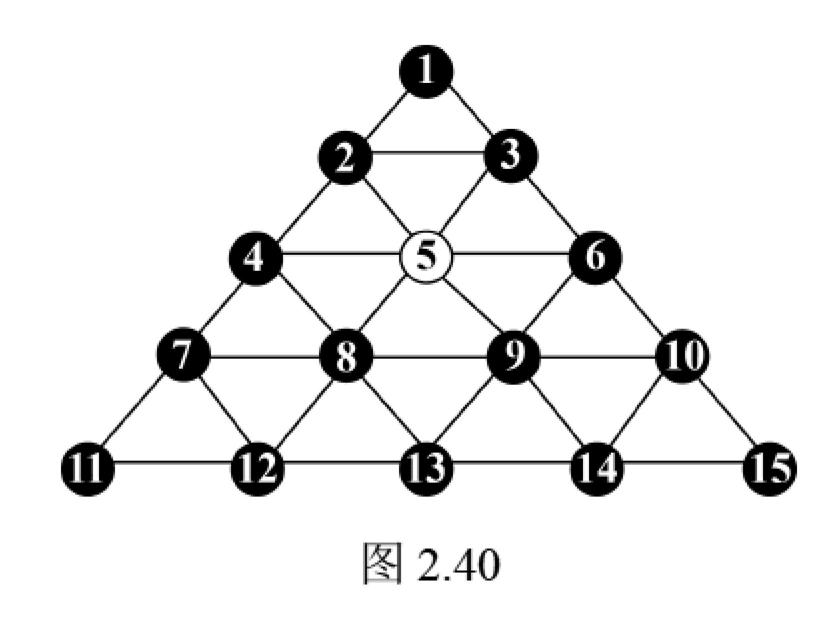


图 2.40

输入仅包含一个整数，即空洞编号，输出最短序列的长度 *m*，然后是 *m* 个整数对，分别表示每次跳跃的小球所在的洞编号以及目标洞的编号。

【分析】

棋盘上小球的跳跃规则比较散乱，可以对每个洞一步就跳到下一个空洞的路径进行编号，按照从小到大的顺序排成一个表格：左上，右上，左，右，左下，右下，如果跳出边界就用 0 来表示。

可以使用一个位向量来表示当前棋盘的局面。同时要记录从初始状态到当前状态的跳跃路径（每一次跳跃都记录起点和目的点）。

最后就是使用 BFS 来对所有的状态进行搜索，每一步可以选择一个球的位置（按照从小到大的顺序来选），然后同样按照从小到大的顺序尝试往 6 个方向跳到最近的空洞。所有合法的跳转都将产生一个新的状态，同时还要使用一个 set<int>对所有已经入队的棋盘局面（前述的二进制表示）进行判重，每一个新局面如果已经入队，就退出。

程序代码（C++11）如下：

```
using namespace std;
int readint() { int x; cin>>x; return x; }

template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v) {
   bool first = true;
   for(const auto& e : v) {
      if(first) first = false; else os<<" ";
      os<<e;
   }
   return os;
}

const int N = 15, D = 6, DIRS[N+1][D] = { //6 个方向能跳到的洞
```

```
    {0,0,0,0,0,0 },
    {0,0,0,0,2,3 },      //1
    {0,1,0,3,4,5 },      //2
    {1,0,2,0,5,6 },      //3
    {0,2,0,5,7,8 },      //4
    {2,3,4,6,8,9 },      //5
    {3,0,5,0,9,10 },     //6
    {0,4,0,8,11,12 },    //7
    {4,5,7,9,12,13 },    //8
    {5,6,8,10,13,14 },   //9
    {6,0,9,0,14,15 },    //10
    {0,7,0,12,0,0 },     //11
    {7,8,11,13,0,0 },    //12
    {8,9,12,14,0,0 },    //13
    {9,10,13,15,0,0 },   //14
    {10,0,14,0,0,0 }     //15
};

struct Board{ //棋盘状态
    int pos, cnt; //表示每个位置上是否有小球的位向量，现有小球的个数
    vector<int> path;
    Board() { pos = ~0; cnt = N; }
    //取出 i 的棋子
    void clear(int i) { assert(test(i)); pos ^= (1<<i); cnt--; }
    //在 i 放棋子
    void put(int i) { assert(!test(i)); pos |= (1<<i); cnt++; }
    //i 位置上有棋子吗
    bool test(int i) const { return (pos & (1<<i)) != 0; }
    int findJump(int i, int d) const { //i 往 d 方向能跳吗，返回是跳的最远位置
        int j = DIRS[i][d];
        if(!j || !test(j)) return 0;
        while(j && test(j)) j = DIRS[j][d];
        return j;
    }

    void dbgPrint() {
        int len = 1, p = 1;
        cout<<"cnt = "<<cnt<<endl;
        for(int i = 1; i <= N; i++) {
            if(test(i)) cout<<"*";
            else cout<<"_";
            if(i == p) { cout<<endl; len += 1; p += len; }
```

```
        }
        cout<<endl<<path.size()<<" -> "<<path<<endl;
    }
};

void solve(int e) {
    Board ib; ib.clear(e);
    queue<Board> q; q.push(ib);
    unordered_set<int> vis; vis.insert(ib.pos);

    while(!q.empty()) {
        Board b = q.front(); q.pop();
        if(b.cnt == 1 && b.test(e)) {
            cout<<b.path.size()/2<<endl;
            cout<<b.path<<endl;
            return;
        }

        for(int i = 1; i <= N; i++) {
            if(!b.test(i)) continue;
            _for(d, 0, D) {                          //尝试不同的方向
                int t = b.findJump(i, d);            //看看最远跳多远
                if(!t) continue;
                int j = DIRS[i][d];                  //assert(j && b.test(j));
                Board nb = b;
                nb.clear(i);                         //从 i 起跳
                while(j != t) nb.clear(j), j = DIRS[j][d]; //路过的棋子全部取出来
                nb.put(t);
                if(vis.count(nb.pos)) continue;   //局面已经搜过
                vis.insert(nb.pos);
                nb.path.push_back(i);
                nb.path.push_back(t);
                q.push(nb);
            }
        }
    }

    cout<<("IMPOSSIBLE\n");
}

int main() {
    int T = readint();
```



```
    while(T--) solve(readint());
    return 0;
}
```

习题 7-15　最大的数（Biggest Number, UVa11882）

在一个 R 行 C 列（$2 \leqslant R$, $C \leqslant 15$，$R*C \leqslant 30$）的矩阵里有障碍物和数字格（包含 1～9 的数字）。你可以从任意一个数字格出发，每次沿着上下左右之一的方向走一格，但不能走到障碍格中，也不能重复经过一个数字格，然后把沿途经过的所有数字连起来。

如图 2.41 所示，你可以得到 9784、4832145 等整数。问：能得到的最大整数是多少？

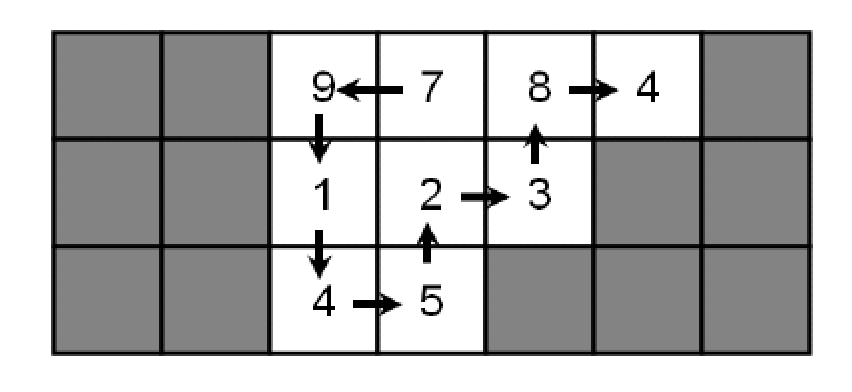


图 2.41

【分析】

直接暴力搜索，就是考虑当前的位置、走过的位置，以及已经走出来的整数作为当前状态，每次走一步，看看后续能走出多大的整数。但是这样极端情况下状态个数约为 4^{30}，肯定会超时。

可以考虑两个剪枝优化：

（1）走到一个点（x,y），当前已经走出的数字序列是 cur，已经搜到的最优答案为 ans。往下搜索之前，看看最多还能走出几步。可以使用 BFS 把跟当前点依然连通的数字搜索出来（不包括已经走过的），记这个数字序列为 rs，那么这个 rs 的长度就是后续能走出的步数上限，如果 cur.size() + rs.size() < ans.size()，说明无论如何走出来的数字长度都不会超过 ans。也就不可能搜出最优解，直接退出。

（2）如果 cur.size() + rs.size() == ans.size()，说明有可能走出更大的数字，那就把 rs 从大到小排序，如果排序后的 cur 和 rs 拼起来小于 ans 对应的数字，直接退出。

☞**注意：**

对应的数字可能超过 30 位，可以用一个 vector<char>来储存，这样数字的修改和数字之间的比较都比较方便。

完整程序如下：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
using namespace std;
typedef long long ll;
const int MAXR = 32, MAXC = 32;
```

```
int R,C, Walked[MAXR][MAXC], Vis[MAXR][MAXC], DX[4] = {0,1,0,-1}, DY[4] =
{1,0,-1,0};
char Grid[MAXR][MAXC];

inline bool isValid(int x, int y) {
    return 0 <= x && x < R && 0 <= y && y < C && isdigit(Grid[x][y]);
}

struct Rec {
    vector<char> buf;
    bool operator<(const Rec& r2) const {
        if(buf.size() != r2.buf.size()) return buf.size() < r2.buf.size();
        return buf < r2.buf;
    }
    Rec& operator+=(const Rec& r2) {
        int sz = buf.size();
        buf.resize(buf.size() + r2.buf.size());
        copy_backward(begin(r2.buf), end(r2.buf), begin(buf) + sz);
        return *this;
    }
    Rec& operator+=(char c) { buf.push_back(c); return *this; }
    inline int size() const { return buf.size(); }
    inline void clear(){ buf.clear(); }
    void printLn() const {
        for(int i = 0; i < size(); i++) putchar(buf[i]);
        puts("");
    }
};

void bfs(int x, int y, Rec& rs) { //已经走到(x,y)，还能走多远，BFS 一下，存到 rs 中
    queue<int> q;
    q.push(x * MAXC + y);
    memset(Vis, 0, sizeof(Vis));
    Vis[x][y] = 1;
    while(!q.empty()) {
        int tmp = q.front(); q.pop();
        x = tmp / MAXC, y = tmp % MAXC;
        _for(i, 0, 4) {
            int ax = x + DX[i], ay = y + DY[i];
            if(!isValid(ax,ay) || Walked[ax][ay] || Vis[ax][ay]) continue;
            Vis[ax][ay] = 1;
            rs += Grid[ax][ay];
```

```
            q.push(ax*MAXC + ay);
        }
    }
}

bool lessThan(const Rec& l1, Rec& l2, const Rec& t) {
    assert(l1.size() + l2.size() == t.size());
    int i = 0,a,b;
    for(i = 0; i < l1.size(); i++) {
        a = l1.buf[i], b = t.buf[i];
        if(a<b) return true;
        if(a>b) return false;
    }

    sort(l2.buf.begin(), l2.buf.end(), greater<char>());
    for(; i < t.size(); i++) {
        a = l2.buf[i-l1.size()], b = t.buf[i];
        if(a<b) return true;
        if(a>b) return false;
    }
    return false;
}

//从[x,y]开始走，已经走出的数字是 cur，目前的最优答案是 ans
void solve(int x, int y, Rec& cur, Rec& ans) {
    Rec rs;
    bfs(x,y,rs);
    if(cur.size() + rs.size() < ans.size()) return;
    if(cur.size()+rs.size()==ans.size() && lessThan(cur, rs, ans)) return;

    _for(i, 0, 4){
        int ax = x + DX[i], ay = y + DY[i];
        if(!isValid(ax, ay) || Walked[ax][ay]) continue;
        cur += Grid[ax][ay];
        Walked[ax][ay] = 1;
        solve(ax, ay, cur, ans);
        cur.buf.pop_back();
        Walked[ax][ay] = 0;
    }

    if(ans < cur) ans = cur;        //更新答案
}
```

```
int main()
{
    Rec ans, cur;
    while(scanf("%d%d", &R, &C) == 2 && R && C) {
        memset(Grid, 0, sizeof(Grid));
        memset(Walked, 0, sizeof(Walked));
        _for(i, 0, R) scanf("%s", Grid[i]);
        ans.clear();
        _for(i, 0, R){
            _for(j, 0, C){
                if(!isdigit(Grid[i][j])) continue;
                cur.clear(), cur += Grid[i][j];        //尝试从[i,j]开始走
                Walked[i][j] = 1;
                solve(i,j, cur, ans);
                Walked[i][j] = 0;
            }
        }
        ans.printLn();
    }

    return 0;
}
```

习题 7-18　推门游戏（The Wall Pusher, UVa10384）

如图 2.42 所示，你从 S 处出发，每次可以往东、南、西、北 4 个方向之一前进。如果前方有墙壁，游戏者可以把墙壁往前推一格。如果有两堵或者多堵连续的墙，游戏者不能将它们推动。另外，游戏者也不能把游戏区域边界上的墙推动。

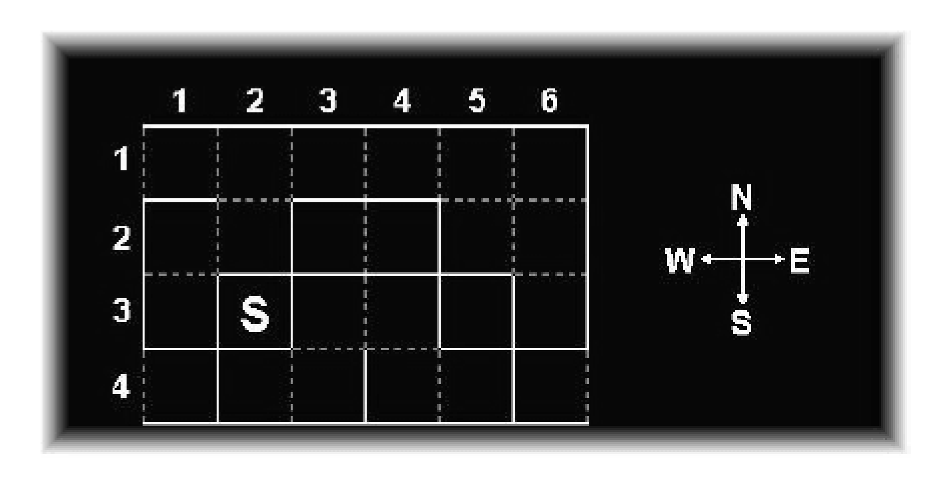


图 2.42

用最少的步数走出迷宫（边界处没有墙的地方就是出口）。迷宫总是有 4 行 6 列，多解时任意输出一个移动序列即可（用 NEWS 4 个字符表示移动方向）。

【分析】

如果使用 BFS，搜索时除了当前的位置，还要考虑墙的状态，结点判重更不好处理。所以考虑使用迭代加深搜索（IDFS）作为主算法框架。

每步有两种可能的走法：

（1）沿着这个方向可以直接走到下一个格子。

（2）可以沿着当前的方向把墙推到下一个格子。

然后就可以从当前位置每一步选择 4 个方向其中之一搜索即可。程序实现上有以下几点需要注意：

（1）使用 0、1、2、3 来表示 WNES 这 4 个方向，然后输出时再转换即可。

（2）使用两个数组来存储 4 个方向的向量：int DX[] = { 0, −1, 0, 1 }，DY[] = { −1, 0, 1, 0 }。

（3）使用一个数组来表示 4 个方向的反方向：REVD[] = { 2, 3, 0, 1 }，用来在推门时给下一个格子去掉对应的墙，详细参见代码。这样遍历 4 个方向的代码就可以统一处理。

完整程序（C++11）如下：

```
using namespace std;

const int R = 4, C = 6;
const string DC = "WNES";                // 0 1 2 3
int DX[] = { 0, -1, 0, 1 }, DY[] = { -1, 0, 1, 0 }, REVD[] = { 2, 3, 0, 1 };
int readint() { int x; cin >> x; return x; }

template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v) {
    for(const T& e : v) os<<e;
    return os;
}
//位运算封装，读写 x 的第 b 位
bool get(int x, int b) { return (x & (1 << b)) > 0; }
void set(int& x, int b, bool v) {
    if (v) x |= (1 << b);
    else x &= ~(1 << b);
}
int Vis[R][C], cells[R][C];
bool IsValid(int x, int y) { return x >= 0 && x < R && y >= 0 && y < C; }
bool isExit(int x, int y, vector<char>& path) {
   int p = cells[x][y];
   if (x == 0 && !get(p, 1)) { path.push_back(DC[1]); return true; }
//第一行
   if (y == 0 && !get(p, 0)) { path.push_back(DC[0]); return true; }
//第一列
   if (x == R - 1 && !get(p, 3)) { path.push_back(DC[3]); return true; }
//最后一行
   if (y == C - 1 && !get(p, 2)) { path.push_back(DC[2]); return true; }
//最后一列
```

```
    return false;
}

//坐标，路径，步数，最大搜索深度
bool dfs(int x, int y, vector<char>& path, int d, const int maxd) {
    assert(IsValid(x, y));
    if (isExit(x,y,path)) return true;
    if (d >= maxd) return false;
     int& p = cells[x][y];
    _for (i, 0, 4) {
        int ax = x + DX[i], ay = y + DY[i];
        if (!IsValid(ax, ay) || Vis[ax][ay]) continue;
        int& np = cells[ax][ay];

       path.push_back(DC[i]);
       Vis[ax][ay] = 1;

        if (!get(p, i)) { //这个方向没有墙
            if (dfs(ax, ay, path, d + 1, maxd)) return true;
        }
        else if (!(get(np, i))){ //有墙，但是可以推过去
            set(p, i, 0);  set(np, i, 1); set(np, REVD[i], 0);
            int aax = ax + DX[i], aay = ay + DY[i];
//推过去之后下一个受墙影响的位置
            if (IsValid(aax, aay)) set(cells[aax][aay], REVD[i], 1);
            if (dfs(ax, ay, path, d + 1, maxd)) return true;
            if (IsValid(aax, aay)) set(cells[aax][aay], REVD[i], 0);
            set(p, i, 1); set(np, i, 0); set(np, REVD[i], 1);
        }

       Vis[ax][ay] = 0;
       path.pop_back();
    }

    return false;
}

int main(){
    int sx, sy, maxd;
    vector<char> path;
    while ((sy = readint()) && (sx = readint())) {
        _for(i, 0, R) _for(j, 0, C) cin>>cells[i][j];
```

```
        sx--; sy--; maxd = 1;
        while (true) {
            memset(Vis, 0, sizeof(Vis));
            path.clear();
            Vis[sx][sy] = 1;
            if (dfs(sx, sy, path, 0, maxd)) break;
            Vis[sx][sy] = 0;
            maxd++;
        }

        cout << path << endl;
    }

    return 0;
}
```

2.6 高效算法设计

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 8 章。

习题 8-1 装箱（Bin Packing, SWERC 2005, UVa1149）

给定 N（$N \leqslant 10^5$）个物品的重量 L_i，背包的容量 M，同时要求每个背包最多装两个物品。求至少要多少个背包才能装下所有的物品。

【分析】

先对 N 个物体按照重量从大到小排序。依次进行决策。在没放好的物体中，考虑最重的 i，它应该和谁放一起呢？选择最轻的 j，如果 $i+j$ 都放不到一个背包里，那 i 只能单独放，否则就把 i 和 j 放在一起。

在考虑 i 时，如果有多个 j 都可以和 i 放在一起，那么选择最轻的 j 是不是最优策略？对于所有比 j 重的 k 来说，在 i 决策完之后考虑的物体，重量小于 i，所以 k 依然可以和 i 之后的物体放在一起，不会多占背包。完整程序如下：

```
const int MAXN = 100000 + 10;
int n, l, len[MAXN];

int readint() { int x; scanf("%d", &x); return x; }

void solve() {
   n = readint(), l = readint();
   _for(i, 0, n) len[i] = readint();
   sort(len, len + n, greater<int>());
```

```
    int ans = 0, left = 0, right = n - 1;
    while(left <= right) { //最左边是当前最重的，最右边是当前最轻的
        ans++; left++;
        if(left != right && len[left] + len[right] <= 1) right--;
    }
    printf("%d\n", ans);
}

int main()
{
    int k = readint(), first = 1;
    while(k--) {
        if(first) first = 0; else puts("");
        solve();
    }
    return 0;
}
```

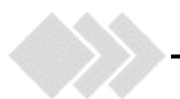
习题 8-2　聚会游戏（Party Games, Mid-Atlantic 2012, UVa1610）

输入一个 n（$2 \leqslant n \leqslant 1000$，$n$ 是偶数）个字符串的集合 D，找一个字符串（不一定在 D 中出现）S，使得 D 中恰好一半串≤S，另一半串>S。如果有多解，输出字典序最小的解。例如，对于{JOSEPHINE, JERRY}，输出 JF；对于{FRED, FREDDIE}，输出 FRED。

【分析】

记 $n=2k$。首先对 D 进行递增排序，所求的字符串 P 一定满足 $D_k \leqslant P < D_{k+1}$，则问题就变成寻找符合这个条件的最短并且字典序最小的字符串。记 D_k 的长度为 L，则 P 的长度一定不大于 L。参考暴力搜索的思路，从 P 的最左边第 0 位到第 L−1 位逐步从小到大尝试构造每一位的字符。对于第 i 位，开始令 P[i]='A'，只要满足 P[i]≤'Z'且 P < D_k，就一直令 P[i]递增。这样构造出来的 P[i]如果满足 P[i]≤'Z'且 $D_k \leqslant P < D_{k+1}$，则说明构造完成，否则继续构造下一位。每一位构造完成之后所得的 P 就是所求的解。完整程序如下：

```
using namespace std;
int main() {
    const int MAXN = 1000 + 4;
    int n; string P, D[MAXN];
    while(cin>>n && n) {
        _for(i, 0, n) cin>>D[i];
        sort(D, D + n);
        const string &l = D[n/2-1], &r = D[n/2];
        P = "A";
        int i = 0, sl = l.size();
        while (i < sl) {
```



```
            while (P[i] <= 'Z' && P < l) ++P[i];
            if (P[i] <= 'Z' && P >= l && P < r) break;
            if (l[i] != P[i]){ assert(P[i] == l[i] + 1); --P[i]; }
            P += 'A';
            ++i;
        }
        cout<<P<<endl;
    }
    return 0;
}
```

习题 8-4 奖品的价值（Erasing and Winning, UVa11491）

你是一个电视节目的获奖嘉宾。主持人在黑板上写出一个 N 位整数（不以 0 开头），并邀请你恰好删除其中的 D 个数字。剩下的整数便是你所得到的奖品的价值。自然地，你希望这个奖品价值尽量大。$1 \leqslant D < N \leqslant 10^5$。

【分析】

令 $E=N-D$，则本题就是要从输入的整数中选择 E 个数字，使得组成的整数最大。可以连续选择 E 次最左边的最大数字，还要给后续的选择留够位数，每次选择的位置为 pos，则下次选择的范围就从 pos+1 开始。

注意有一个子操作：从一个区间内选择最左边的最大数字。因为 N 和 D 的规模都比较大，如果每次线性查找，最坏情况下的时间复杂度会达到 $O(N^2)$级别，对于输入的规模是无法接受的[①]。可以做如下的预处理：对于每个位置 i 以及 $d \in [0, 9]$，记录在区间[i…]内第一个 d 出现的位置 NEXT[i][d]。这个预处理可以在 $O(10n)$的时间内完成，然后上述子操作就可以在 $O(9)$的时间内完成。

完整程序如下：

```
using namespace std;
int readint() { int x; scanf("%d", &x); return x; }

const int MAXN = 100000 + 4;
int NUM[MAXN], NEXT[MAXN][10];

void init(int N) {
    _for(d, 0, 10){
        int pos = N;                //目前为止出现的最左边的 d 的位置
        for(int j = N - 1; j >= 0; j--) {
            if(NUM[j] == d) pos = j;
            NEXT[j][d] = pos;
        }
```

① 具体请参考《算法竞赛入门经典（第 2 版）》中 8.1.4 节中的表 8-1。

```
    }
}

//在NUM的[start, end)区间内选最左边的最大值
int select_max(int start, int end, int& pos) {
    for(int d = 9; d >= 0; d--) {
        if(NEXT[start][d] < end) {
            pos = NEXT[start][d];
            return d;
        }
    }
    assert(false);
}
//要在NUM[0...N]中选择E位的最大数字
void solve(int N, int E) {
    init(N);
    string ans;
    int start = 0;
    while(E--) {
        int pos;
        ans += select_max(start, N-E, pos) + '0';
        start = pos + 1;
    }
    puts(ans.c_str());
}

char buf[MAXN];
int main()
{
    int N, D;
    while(scanf("%d%d\n", &N, &D) == 2 && N && D){
        gets(buf);
        _for(i, 0, N) NUM[i] = buf[i]-'0';
        solve(N, N - D);
    }
    return 0;
}
```

习题 8-5　折纸痕（Paper Folding, UVa177）

你喜欢折纸吗？给你一张很大的纸，对折以后再对折，再对折……每次对折都是从右往左折，因此在折了很多次以后，原先的大纸会变成一个窄窄的纸条。现在把这个纸条沿着折纸的痕迹打开，每次都只打开“一半”，即把每个痕迹做成一个直角，那么从纸的一



端沿着和纸面平行的方向看过去，会看到一个美妙的曲线，如图 2.43 所示。

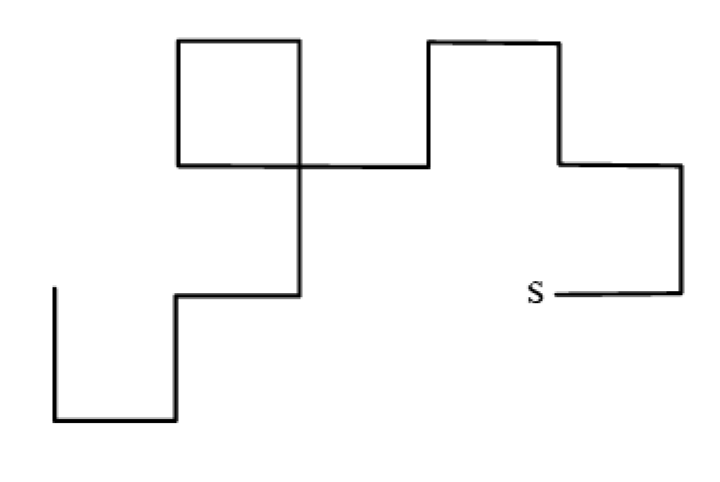


图 2.43

例如，如果你对折了 4 次，那么打开以后你将看到如图 2.43 所示的曲线。注意，该曲线是不自交的，虽然有两个转折点重合。给出对折的次数，请编程绘出打开后生成的曲线。

【分析】

看到题目很多读者应该会和笔者一样，首先想到去直接模拟折叠过程。

但经过思考可以发现一个关键点：可以忽略折叠的过程。直接从一条水平的单位长度的线段开始张开，每次张开都是把已有的所有线段首先围绕一个点顺时针旋转 90°，然后再和旋转之前的线段一起组合成为新的图形。每次旋转的过程中，需要维护两个点的坐标：起始点 s 和旋转中心点 rot。每次旋转，原先的 s 不变，s 经过旋转之后的那个点成为新的 rot，如图 2.44 所示。

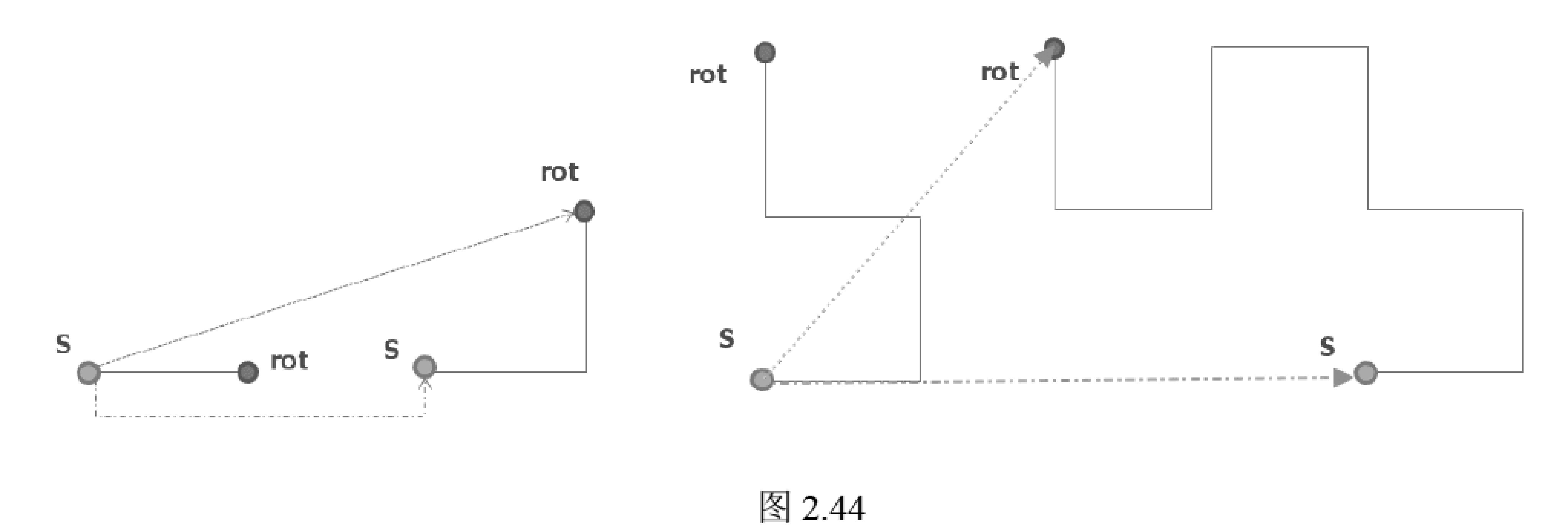


图 2.44

旋转的次数就是输入的 *n*。

当把所有最终线段的坐标模拟出来之后，就剩下输出的问题。有一种比较简便的处理方法就是扫描所有的线段，将其 x 坐标放大一倍，然后垂直类型线段的 x 坐标再减 1。判断平面上每个点是否有垂直或者水平线段，记录对应的字符。最后再扫描平面上的每个点，输出之前记录的字符即可。

完整程序如下：

```
//p 围绕 r 顺时针旋转 90°，返回旋转后的点
Point rotate(const Point& p, const Point& r) {
   Vector pv = p - r;
   Point ans = r + Vector(pv.y, -pv.x);
   return ans;
}
```

```
const int MAXN = 13;
struct Line {
    Point start, end;
    bool vertical;

    //围绕 r 顺时针旋转 90°
    Line rotate(const Point& r) {
        Line ret;
        ret.start = ::rotate(start, r);
        ret.end = ::rotate(end, r);
        return ret;
    }
    //规整，保证 start 在 end 的左边或者上边
    void normalize() {
        assert(start != end);
        assert(start.x == end.x || start.y == end.y);
        vertical = (start.x == end.x);
        if(vertical) {
            if(start.y > end.y) swap(start.y, end.y);
        } else {
            if(start.x > end.x) swap(start.x, end.x);
        }
    }
};

int n, LineCnt;
vector<Line> lines;

int main()
{
    lines.reserve(1<<MAXN);
    while(cin>>n&&n) {
        Line l;
        l.end = Point(1, 0);
        l.vertical = false;
        lines.clear();
        lines.push_back(l);
        int maxY = l.start.y, minY = l.start.y,
            minX = l.start.x, maxX = l.end.x;

        Point s = l.start, rot = l.end;
```

```
        _for(i, 0, n) {
            int sz = lines.size();
            _for(j, 0, sz) {
                Line nl = lines[j].rotate(rot);
                nl.normalize();
                lines.push_back(nl);
            }
            rot = rotate(s, rot);
        }

        map<Point, char> pc;
        for(auto& l : lines) {
            Point& lp = l.start;
            lp.x *= 2;
            if(l.vertical) lp.x--;
            minX = min(lp.x, minX);
            maxX = max(lp.x, maxX);
            minY = min(lp.y, minY);
            maxY = max(lp.y, maxY);
            pc[lp] = l.vertical ? '|' : '_';
        }

        // cout<<"minX = "<<minX<<endl;
        string buf;
        for(int y = maxY; y >= minY; y--) {
            buf.clear();
            for(int x = minX; x <= maxX; x++) {
                Point p(x,y);
                if(pc.count(p)) buf += pc[p];
                else buf += ' ';
            }
            while(*(buf.rbegin()) == ' ') buf.erase(buf.size()-1);
            cout<<buf<<endl;
        }

        cout<<"^"<<endl;
    }
    return 0;
}
```

习题 8-6　起重机（Crane, ACM/ICPC CERC 2013, UVa1611）

输入一个 1～n（1≤n≤10000）的排列，用不超过 9^6 次操作把它变成升序。每次操作

都可以选一个长度为偶数的连续区间，交换前一半和后一半。如输入 5, 4, 6, 3, 2, 1，可以执行 1, 2 先变成 4, 5, 6, 3, 2, 1，然后执行 4, 5 变成 4, 5, 6, 2, 3, 1，然后执行 5, 6 变成 4, 5, 6, 2, 1, 3，然后执行 4, 5 变成 4, 5, 6, 1, 2, 3，最后执行操作 1,6 即可。

提示：

2*n* 次操作就足够了。

【分析】

这道题目要求排序，但是基本操作却是“交换一个偶数长度的连续区间的前后两半”，依然可以使用冒泡排序的思路，依次把 1 到 *n* 的每个数归位。

遍历到数字 *i* 时，此时 1～*i*−1 已经排好序，在未排序的区间（长度是 *n*−*i*+1）中，记 *i* 前面的元素个数为 *ci*，则有以下两种情况。

（1）*ci*≤(*n*−*i*+1)/2，那么可以通过将 *i* 前面的未排序区间（记其长度为 *ci*）和从 *i* 开始长度为 *ci* 的区间进行交换，即可把 *i* 交换到第 *i* 个位置。

（2）否则，如果 *n*−*i*+1 是偶数，交换前后两半，然后按照情况 1 处理即可。如果 *n*−*i*+1 是奇数，则忽略第一个元素（肯定不是 *i*），交换剩下长度为偶数的区间的前后两半。

这样在≤2*n* 次操作之内就可以完成。

对于输入数据(5 4 6 3 2 1)，算法的运行过程示意图如下，其中灰色表示未排序区间。

i = 1：5 4 6 3 2 1→3 2 1 5 4 6→1 5 3 2 4 6

i = 2：1 5 3 2 4 6→1 2 4 5 3 6

i = 3：1 2 4 5 3 6→1 2 3 6 4 5

i = 4：1 2 3 6 4 5→1 2 3 4 6 5

i = 5：1 2 3 4 6 5→1 2 3 4 5 6

完整程序如下：

```
using namespace std;

int readint() { int x; cin>>x; return x; }

vector<int> C;
typedef vector<int>::iterator TIter;
void dbgPrint(TIter first, TIter last) { while(first != last) cout<<*
(first++)<<","; }
void swapSeg(vector<int>& rec, TIter first, TIter last) {
    int SZ = last - first;
    assert(SZ%2 == 0);
    SZ /= 2;
    rec.push_back(distance(C.begin(), first)+1);
    rec.push_back(distance(C.begin(), last));
    for(int i = 0; i < SZ; i++) swap(*(first+i), *(first+SZ+i));
}
```

```
//处理[first,last)区间变成升序，下一步将其中的数 k 归位，返回所用的交换次数
int solve(vector<int>& rec, int k, TIter first, TIter last) {
    if(first == last) return 0;
    if(*first == k) return solve(rec, k+1, first+1, last);

    int ans = 0;
    TIter kit = find(first, last, k);
    int kp = kit - first, SZ = last - first;
    if(kp > SZ / 2) {
        TIter sLeft = first;
        if(kp%2 == 0) sLeft++;
        swapSeg(rec, sLeft, kit+1);
        ans++;
        kit = find(first, last, k);
        kp = kit - first;
    }

    assert(kp <= SZ/2);
    swapSeg(rec, first, first + (kp*2));
    ans++;
    assert(*first == k);
    return ans + (solve(rec, k+1, first+1, last));
}

int main(){
    int T = readint();
    while(T--) {
        int n = readint();
        C.clear();
        while(n--) C.push_back(readint());
        vector<int> rec;
        int m = solve(rec, 1, C.begin(), C.end());
        assert(m == rec.size()/2);
        cout<<m<<endl;
        for(int i = 0; i < rec.size(); i += 2) cout<<rec[i]<<" "<<rec[i+1]<<endl;
    }
    return 0;
}
```

习题 8-8 猜名次（Guess, ACM/ICPC Beijing 2006, UVa1612）

有 n（$n \leqslant 16384$）位选手参加编程比赛。比赛有 3 道题目，每个选手的每道题目都有一个评测之前的预得分（这个分数和选手提交程序的时间相关：提交得越早，预得分越大）。

接下来是系统测试。如果某道题目未通过测试，则该题的实际得分为 0 分，否则得分等于预得分。得分相同的选手，ID 小的排在前面。

给出所有 3*n* 个得分以及最后的实际名次，问是否可能。如果可能，输出最后一名的最高可能得分。每个预得分均为小于 1000 的非负整数，最多保留两位小数。

【分析】

考虑每个选手，每道题目的得分有两种可能（预得分或 0），那么总得分就是 $2^3=8$ 种可能。输入时，就计算每个选手的这 8 种得分并且从大到小排序。

第一名的得分选择为 8 个得分中的最大值，然后按照名次从前到后遍历每个选手，从其 8 个可能得分中选择满足以下条件的最大得分：

（1）小于上个选手的得分。

（2）或者等于上一个选手的得分且 ID 小于其 ID。

如果选择成功，则这个得分必定是当前选手符合输入名次的最大可能得分。如果失败，说明这个名次无法构造，直接退出。所有选手选择成功后，直接输出最后一个选手选择的得分。

需要注意的是，虽然分数是浮点数，但是题目标明了只有小数点后两位。所以可以直接乘以 100 转换为整数，最后输出时再除以 100。为了避免转换误差，在输入时作为字符串输入，然后再读取为两个 int。输出时做类似的处理。完整程序（C++11）如下：

```
using namespace std;
int readint() { int x; scanf("%d", &x); return x; }
const int maxn = 16384 + 4;
vector<int> PS[maxn], IDs;
int n;

int solve() {
   int lastScore, lastId = -1;
   for (auto id : IDs) {
      const auto& p = PS[id];
      if (lastId == -1)
         lastScore = p.front();
      else
      {
         bool found = false;
         for(auto s : p){
            if (s < lastScore || (s == lastScore && id > lastId)) {
               lastScore = s;
               found = true;
               break;
            }
         }
```

```
            if (!found) return -1;
        }
        lastId = id;
    }
    return lastScore;
}

int readF2i() {
    char buf[8];
    scanf("%s", buf);
    int a, b = 0;
    sscanf(buf, "%d", &a);
    char *pp = strchr(buf, '.');
    if (pp) sscanf(++pp, "%d", &b);
    return a*100 + b;
}

int main(){
    int n;
    for (int k = 1; scanf("%d", &n) == 1 && n; k++) {
        _for(i, 0, n) {
            auto& p = PS[i];
            p.clear(); p.push_back(0);
            _for (j, 0, 3) p.push_back(readF2i());
            p.push_back(p[1]+p[2]),   p.push_back(p[1]+p[3]),   p.push_back
(p[2]+p[3]);
            p.push_back(p[1]+p[2]+p[3]);
            sort(begin(p), end(p), greater<int>());
        }
        IDs.clear(); _for (i, 0, n) IDs.push_back(readint() - 1);
        int ans = solve();
        if (ans == -1) printf("Case %d: No solution\n", k);
        else printf("Case %d: %d.%02d\n", k, ans / 100, ans % 100);
    }
    return 0;
}
```

习题 8-11　高速公路（Highway, ACM/ICPC SEERC 2005, UVa1615）

给定平面上 n（$n \leqslant 10^5$）个点和一个值 D，要求在 x 轴上选出尽量少的点，使得对于给定的每个点，都有一个选出的点离它的欧几里得距离不超过 D。

【分析】

对于每个指定的点 P，X 轴上符合要求的点刚好组成 X 轴和圆（P, D）相交的那根弦所

对应的区间。那么题目就转换为，在一系列的区间内选择最少的点，使得每个区间内都有点被选中。具体的思路可参考《算法竞赛入门经典（第 2 版）》中的第 8.4.2 节。

习题 8-14　商队抢劫者（Caravan Robbers, ACM/ICPC NEERC 2012, UVa1616）

输入 n 条线段，把每条线段变成原线段的一条子线段，使得变之后所有线段等长且不相交（但是端点可以重合）。输出最大长度（用分数表示）。例如，有 3 条线段[2,6]、[1,4]、[8,12]，则最优方案是分别变成[3.5,6]、[1,3.5]、[8,10.5]，输出 5/2。

【分析】

首先把所有线段按照左端点从小到大进行排序，然后使用二分法来计算子线段的最大长度。对于长度 L，针对每一个原线段[a,b]，尽量选择靠左的区间作为子线段，如果子线段的左端点 max(a,lb) + L > b，那么选择失败，其中 lb 是上一个选择的子线段的右端点。如果每个线段选择成功，那么 L 有效。

但是需要注意以下几点：

（1）二分开始要选择尽量窄的一个起始区间，可以选择为[1, 1000000/n]。

（2）二分的迭代次数 50 次就足够，无须将区间缩小到足够小的范围，否则会超时。

（3）选择输出目标的有理数时，遍历所有可能的分母（就是 1～n），然后根据分母以及算出的长度来选择分子。最后选择误差最小的分子分母组合来输出，输出之前都要除去二者的最大公约数。

完整程序如下：

```
using namespace std;

int gcd( int a, int b ) { return b ? gcd(b, a % b) : a; }

const double eps = 1e-7;
int dcmp(double x) { if(fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
int dcmp(double x, double y) { return dcmp(x-y); }

struct Seg{
   int a, b;
   bool operator<(const Seg& s) const { return a <= s.a; }
};

const int maxn = 100000 + 4;
int n;
Seg segs[maxn];

bool tryLen(const double l) {
   double lb = 0;
   _for(i, 0, n){
      const Seg& s = segs[i];
```

```
        lb = max((double)s.a, lb) + 1;
        if(lb > s.b) return false;
    }
    return true;
}

void output(double l) {
    double ip = floor(l + eps);
    if(dcmp(ip, l) == 0) {
        printf("%.0lf/1\n", ip);
        return;
    }
    //p/q
    int p=1, q=1;
    double ans = 1;                                    //目前为止最接近 l 的 p/q 值
    for(int i = 1; i <= n; i++) {                      //i 作为分母
        int cp = (int)(floor(l*(double)i + 0.5));      //可能的分子
        double x = (double)cp/i;
        if(fabs(x - l) < fabs(ans - l)) {
            q = i;
            p = cp;
            ans = x;
        }
    }

    int g = gcd(p,q);
    printf("%d/%d\n", p/g, q/g);
}

int main() {
    while(scanf("%d", &n) == 1) {
        _for(i, 0, n) {
            Seg& s = segs[i];
            scanf("%d%d", &s.a, &s.b);
        }
        sort(segs, segs + n);
        double l = 1, r = (double)1000000.0/n, m;
        assert(dcmp(l,r) <= 0);
        _for(b, 0, 50){
            m = (l+r)/2;
            if(!tryLen(m)) r = m;
            else l = m;
```

```
        }
        output((l+r)/2);
    }
    return 0;
}
```

习题 8-16　弱键（Weak Key, ACM/ICPC Seoul 2004, UVa1618）

给 k（$4 \leqslant k \leqslant 5000$）个整数组成的序列 N_i，判断是否存在 4 个整数 N_p、N_q、N_r 和 N_s（$1 \leqslant p<q<r<s \leqslant k$），使得 $N_q>N_s>N_p>N_r$ 或者 $N_q<N_s<N_p<N_r$。

【分析】

首先对输入序列做预处理，对于每个 N_i，记录两个序列 H_i 和 L_i。H_i 包含所有的 j，$j > i$ 且 $N_j > N_i$。L_i 包含所有 j，j 满足 $j > i$ 且 $N_j < N_i$。

然后就是遍历所有的 p，依次选择可能的 q、r、s。

以 $N_q < N_s < N_p < N_r$ 为例：

（1）q 肯定在 L_p 中，在其中进行遍历。

（2）p、q 确定之后，r 肯定在 H_p 中并且 $r > q$，使用 upper_bound 在 H_p 中查找所有可能的 r。

（3）p、q、r 确定之后，s 肯定在 H_q 以及 L_p 中，并且 $s > r$，使用 upper_bound 查找结合 binary_search 判断。

依次进行查找，如果查找到 s 说明有解。查找的过程中判断数字是否在某个序列中并且大于一个数，都可以使用二分查找。可以使用 STL 中的 binary_search 和 upper_bound。完整程序如下：

```
using namespace std;

int readint() { int x; cin>>x; return x; }
const int maxk = 5000 + 4;

vector<int> H[maxk], L[maxk];
int k, N[maxk];

//Nq > Ns > Np > Nr
//p
//q N[q] > N[p] q in H[p]
//r N[r] < N[p] r in L[p] && r > q
//s N[s] > N[p] && N[s] < N[q] && s > r, s in H[p] && s in L[q] && s > r
bool find1(int p) {
    for(auto q : H[p]) {     //Nq > Np, && q > p
        auto rit = upper_bound(L[p].begin(), L[p].end(), q);
        if(rit == L[p].end()) continue;
        int r = *rit;        //Nr < Nq, r > q
```

```
        assert(r > q);
        auto sit = upper_bound(H[p].begin(), H[p].end(), r);
        while(sit != H[p].end()) {
            int s = *sit++;
            assert(s > r);
            if(binary_search(L[q].begin(), L[q].end(), s)) return true;
        }
    }

    return false;
}

//Nq < Ns < Np < Nr
//p
//q q in L[p]
//r r > q && r in H[p]
//s s > r && s in H[q] && s in L[p]
bool find2(int p) {         //Nq > Ns > Np > Nr p<q<r<s
    for(auto q : L[p]){     //Nq > Np, && q > p
        auto rit = upper_bound(H[p].begin(), H[p].end(), q);
        if(rit == end(H[p])) continue;
        int r = *rit;       //Nr < Nq, r > q
        assert(r > q);
        auto sit = upper_bound(L[p].begin(), L[p].end(), r);
        while(sit != L[p].end()) {
            int s = *sit++;
            assert(s > r);
            if(binary_search(H[q].begin(), H[q].end(), s)) return true;
        }
    }

    return false;
}

bool solve() {
    _for(i, 0, k) if(find1(i) || find2(i))  return true;
    return false;
}

int main()
{
    int T = readint();
```

```
    while(T--) {
        k = readint();
        _for(i, 0, k){
            cin>>N[i];
            H[i].clear(), L[i].clear();
        }

        _for(i, 0, k){
            _for(j, i+1, k){
                if(N[j] > N[i]) H[i].push_back(j);
                else if(N[j] < N[i]) L[i].push_back(j);
            }
        }

        if(solve()) cout<<"YES"<<endl;
        else cout<<"NO"<<endl;
    }
    return 0;
}
```

习题 8-17　最短子序列（Smallest Sub-Array, UVa11536）

有 n（$n\leqslant10^6$）个 0～m−1（$m\leqslant1000$）的整数组成一个序列。输入 k（$k\leqslant100$），你的任务是找一个尽量短的连续子序列（$x_a, x_{a+1}, x_{a+2}, \cdots, x_{b-1}, x_b$），使得该子序列包含 1～$k$ 的所有整数。

例如 n=20，m=12，k=4，序列为 1 (2 3 7 1 12 9 11 9 6 3 7 5 4) 5 3 1 10 3 3，括号内部分是最优解。如果不存在满足条件的连续子序列，输出“sequence nai”（不含引号）。

【分析】

使用滑动区间的思路，维护一个区间[L,R]，以及一个 map，其中包含了[L,R]所有 1～k 的整数，及其出现的次数。需要注意的是，map 中只插入 1～k 的整数。则当 map 大小为 k 时[L,R]就是一个符合要求的区间。

首先 L = 0，不断增加 R，直到 map 的大小等于 k 为止，当 map 的大小为 k 时，只要 L 对应的整数 x 不在 1～k 的范围内或者 x 在 map 中出现的次数大于 1，就可以安全地从区间中删除 x，如此反复，当无法继续删除 L 时，就是一个潜在的最小区间。

当发现一个最小区间之后，就把 L 加 1，让区间向右滑动，然后再寻找小区间。如此在滑动的过程中记录下出现的最短区间。完整程序如下：

```
using namespace std;
const int MAXN = 1000001;
int N, M, K, x[MAXN];
int readint() { int x; scanf("%d", &x); return x; }
int in_range(int i) { return i >= 1 && i <= K; }
```

```
int safe_insert(int i, map<int,int>& s) {
    if(in_range(i)) { s[i] = s[i] + 1; }
    return s.size();
}
void safe_del(int i, map<int,int>& s) {
    if(!s.count(i)) return;
    assert(s[i] > 0);
    s[i] = s[i] - 1;
    if(s[i] < 1) s.erase(i);
}

int solve() {
    int ans = 0, L = 0, R = 0;
    _for(i, 0, N) {
        if(i < 3) x[i] = i + 1;
        else x[i] = (x[i-1] + x[i-2] + x[i-3]) % M + 1;
    }

    map<int, int> s;
    safe_insert(x[R], s);
    while(L < N && R < N) {
        while(s.size() < K) {
            safe_insert(x[++R], s);
            if(R >= N) break;
        }

        if(s.size() == K) {
            while(!s.count(x[L]) || s[x[L]] > 1) safe_del(x[L++], s);
            int len = R-L+1;
            if(ans) ans = min(ans, len);
            else ans = len;
        }

        safe_del(x[L], s);
        L++;
        R = max(L, R);
    }

    return ans;
}

int main() {
```

```
    int T = readint();
    _for(t, 1, T+1) {
        printf("Case %d: ", t);
        scanf("%d%d%d", &N, &M, &K);
        int ans = solve();
        if(ans) printf("%d\n", ans);
        else printf("sequence nai\n");
    }

    return 0;
}
```

习题 8-18 感觉不错（Feel Good, ACM/ICPC NEERC 2005, UVa1619）

给一个长度为 n（$n \leqslant 100000$）的非负整数序列 a_i，求出一段连续子序列 $a_l,\cdots a_r$，使得 $(a_l+\cdots+a_r)*\min\{a_l,\cdots,a_r\}$尽量大。

【分析】

对于一个区间[l, r]来说，我们称所求的值为它的权值。假如 a_{r+1} 不是[l, r+1]区间的唯一最小值，那么[l, r+1]的权值一定大于[l, r]的。对于每个 i，假如能让 a_i 成为最小值的最大区间是[l_i, r_i]，则只需要对所有的[l_i, r_i]求权值比较即可。

首先需要预处理出以下数组：

（1）A 的前缀和数组 S，其中 $S_i = \sum_{k=1}^{i} A_i$ 。

（2）L 和 R 数组，其中 L_i 表示 i 左边离 i 最近且比 a_i 小的元素位置。R_i 就是 i 右边离 i 最近的比 a_i 小的元素。让 a_i 成为最小值的最大连续区间就是[L_i, R_i]。

其中第二个预处理的单调栈算法如下（以 L_i 为例）。

首先，在数组 A 前后各附加 1 个 0。使用一个栈 S，初始为空。从左到右把所有下标 i=1～n 入栈，每个下标 i 入栈之前，首先把所有 $a_j \geqslant a_i$ 的下标 j 出栈，之后，栈顶就是左边最接近并且小于 a_i 的元素下标，也就是 L_i 的值。处理完即可得到 L 数组。

以 A = {3,1,6,4,5,2}为例，演示此算法的运行过程。

初始 S = {}，A = {0,3,1,6,4,5,2,0}

i = 1：A[1] = 3, L[1] = 0, S = {1}

i = 2：A[2] = 1, L[2] = 0, S = {2}

i = 3：A[3] = 6, L[3] = 2, S = {2,3}

i = 4：A[4] = 4, L[4] = 2, S = {2,4}

i = 5：A[5] = 5, L[5] = 4, S = {2,4,5}

i = 6：A[6] = 2, L[6] = 2, S = {2,6}

再用类似的逻辑从右到左处理一次即可得到 R 数组。预处理完成后，计算所有 $a_i*(S_{L_i+1}-S_{R_i})$的最大值即可。需要注意的是，可能会有一种特殊情况就是全部元素为 0，则所求结果就是 0，所在区间为[1,1]，需要进行特殊处理。完整程序（C++11）如下：

```
using namespace std;
typedef long long LL;
const int maxn = 100000 + 4;
LL A[maxn], Sum[maxn], L[maxn], R[maxn];
//L[i],R[i]为i的左右两边最接近i的比A[i]小的元素的下标

int main(){
    int n;
    vector<int> s;
    bool first = true;
    while(scanf("%d", &n) == 1){
        if(first) first = false; else puts("");
        s.clear();
        Sum[0] = A[0] = A[n+1] = 0;
        bool allZero = true;
        _rep(i, 1, n){
            scanf("%lld", &(A[i]));
            if(A[i]) allZero = false;
            Sum[i] = A[i] + Sum[i-1];
        }
        Sum[n+1] = Sum[n];

        if(allZero){ printf("0\n1 1\n"); continue; }

        auto popAll = [&s](int i){
            while(!s.empty() && A[s.back()] >= A[i]) s.pop_back();
        };

        _rep(i, 1, n){
            popAll(i);
            L[i] = s.empty() ? 0 : s.back();
            s.push_back(i);
        }
        s.clear();
        _for(i, 0, n){
              int j = n-i; popAll(j);
            R[j] = s.empty() ? n+1 : s.back();
            s.push_back(j);
        }

        LL ans = -1, l = -1, r = -1;
        _rep(i, 1, n){
```

```
            LL li = L[i], ri = R[i]-1, t = A[i] * (Sum[ri]-Sum[li]);
            //printf("i = %d, a = %d, (%d, %d]\n, t = %d", i, A[i], L[i], R[i]-1);
            li++;
            bool update = false;
            if(t < ans) continue;
            if(t > ans) update = true;
            else if(t == ans) {
                if(r - l > ri - li) update = true;
                else if(r - l == ri - li) update = li < l;
            }
            ans = max(ans, t);
            if(update) l = li, r = ri;
        }

        printf("%lld\n%lld %lld\n", ans, l, r);
    }

    return 0;
}
```

习题 8-19　球场（Cricket Field, ACM/ICPC NEERC 2002, UVa1312）

一个 $W*H$（$1 \leqslant W,H \leqslant 10000$）网格里有 n（$0 \leqslant n \leqslant 100$）棵树，要求找一个最大空正方形，如图 2.45 所示。

【分析】

注意网格的长宽都很大，直接枚举正方形的边界（时间复杂度为 10000^3）会超时。但是树只有最多 100 棵，且正方形内部肯定不能包含树。另外，最大正方形一定是有至少两条边都经过一棵树（包括网格边界），如图 2.46 所示，否则很容易构造出更大的正方形。

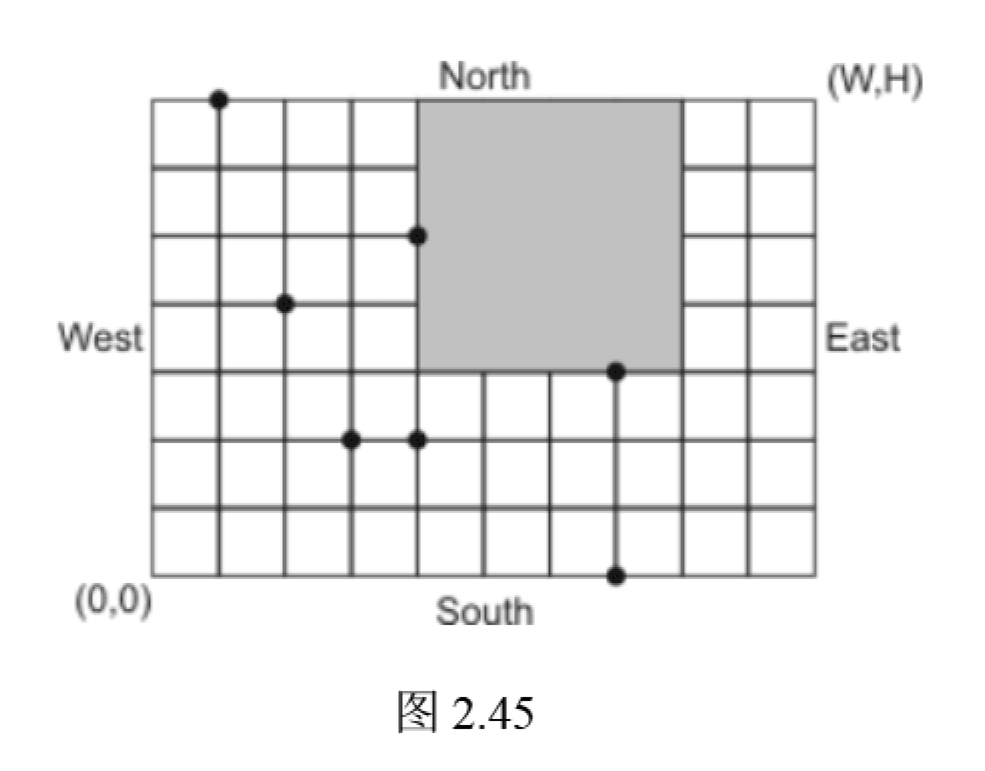
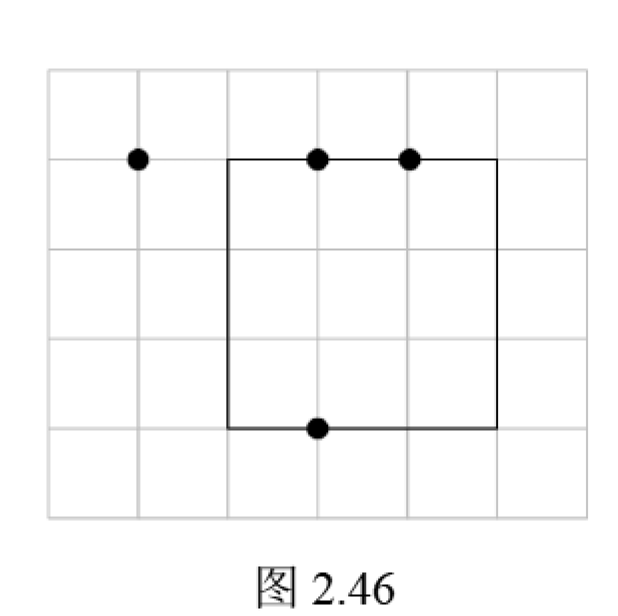


图 2.45


图 2.46

遍历正方形的边界。首先对所有树的 Y 坐标排序去重（使用 set 即可），得到一个正方形可能的上下边的 Y 坐标集合，再对所有树的坐标按照 X 进行排序。

枚举正方形的上下边的 Y 坐标。对于每一对 Y 坐标（minY, maxY），初始的正方形左边界为 left = 0（操场左边）。对于所有的树坐标 P，如果 P.y 恰好正在枚举的上下边界之间，

那么就发现一个新的正方形位于（left, P.x）以及（minY, maxY）相交形成的区域内，更新答案。同时更新 P.x 为下一个正方形的左边界。时间复杂度是 $O(n^3)$。注意整个网格的边界也可以理解为树，不要忘记处理。完整程序如下：

```
using namespace std;

int readint() { int x; cin>>x; return x; }

const int MAXN = 104;
struct Point {
   int x, y;
   bool operator<(const Point & p) const { return x < p.x; }
};
istream& operator>>(istream& is, Point& p) { return is>>p.x>>p.y; }

Point Ps[MAXN];
int N, W, H;
set<int> Ys;                        //所有的树的 Y 坐标

int solve(Point& ans) {
   sort(Ps, Ps+N);
   int len = 0;
   vector<int> Y(Ys.begin(), Ys.end());
   _for(a, 0, Y.size()) _for(b, a+1, Y.size()) {
      int miny = Y[a], maxy = Y[b], dy = maxy - miny;
      if(dy <= len) continue;      //边长

      int left = 0;                 //正方形左边界
      _for(i, 0, N) {
         const Point& p = Ps[i];
         if(p.y <= miny || p.y >= maxy) continue;
         if(len < min(dy, p.x - left)) {
            len = min(dy, p.x - left);
            ans.x = left;
            ans.y = miny;
         }
         left = p.x;
      }
      if(len < min(dy, W - left)) {
         len = min(dy, W - left);
         ans.x = left;
```

```
            ans.y = miny;
        }
    }

    return len;
}

int main(){
    int T = readint();
    Point ans;
    _for(t, 0, T) {
        if(t) cout<<endl;
        cin>>N>>W>>H;
        Ys.clear(), Ys.insert(0), Ys.insert(H);
        _for(i, 0, N) cin>>Ps[i], Ys.insert(Ps[i].y);
        int len = solve(ans);
        cout<<ans.x<<" "<<ans.y<<" "<<len<<endl;
    }
    return 0;
}
```

习题 8-24　龙头滴水（Faucet Flow, UVa10366）

x=0 的正上方有一个水龙头，以每秒 1 单位体积的速度往下滴水。*x*=−1, −3, …, leftx 和 *x*=1, 3, 5, …, rightx 处各有一个挡板，高度已知。求经过多长时间以后水会流出最左边的挡板或者最右边的挡板。如图 2.47 所示，leftx=−3，rightx=3，4 个挡板高度分别为 4、3、2、1，则 6 秒钟之后水会从最右边的挡板溢出。

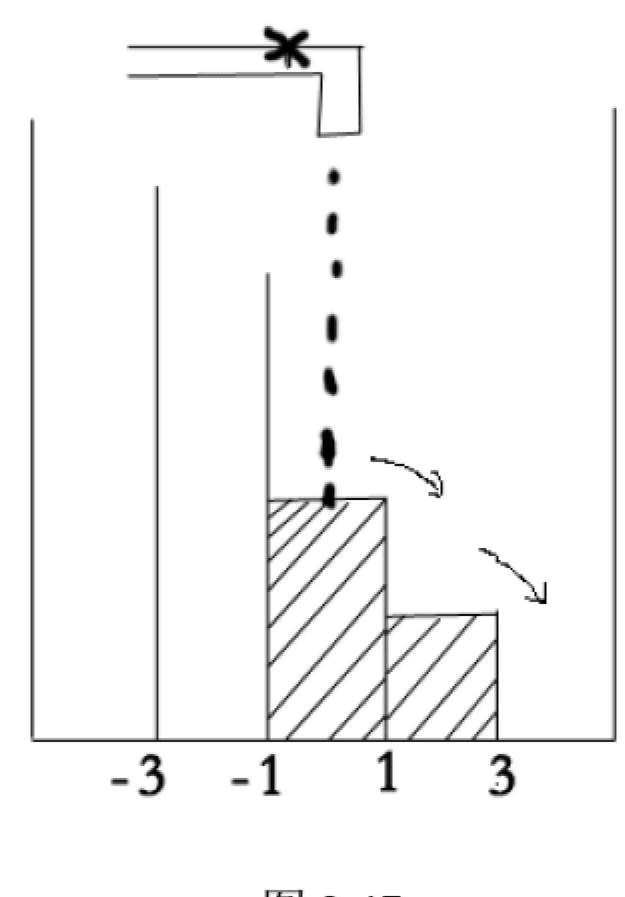


图 2.47

输入第一行为两个奇数 leftx 和 rightx（leftx≤-1，rightx≥1），接下来的各个正整数表示从左到右各个挡板的高度。挡板个数不超过 1000。

【分析】

首先引入一个关键的结论：

如果有两个挡板 X 和 Y（如图 2.48 所示），X 不高于 Y，那么从 X 左边的水如果要流到 Y，在接触到 Y 之前，会形成一个阶梯形状。也就是说，从 X 流到 Y 所需要的时间就是阶梯下方的面积。

回到题目本身，考虑 X = 0 的左右两边最高的挡板高度 LH、RH，以及其位置 LHi、RHi（如果有多个，就取离 X=0 最近的那个）。

如果 LH = RH，那么说明水流会在两边都溢出来。那么就计算水从 LHi 流到最左边挡板所需要的时间 Lt，以及从 RHi 流到最右边挡板所需要的时间 Rt。因为两边同时在流，所求的结果就是水把 LHi-RHi 这个区间的矩形装满所需要的时间再加上 min(Lt +Rt)*2。

如果 LH < RH，假设 Ti 是 X=0 右边第一个高度≥LH 的，如图 2.49 所示。

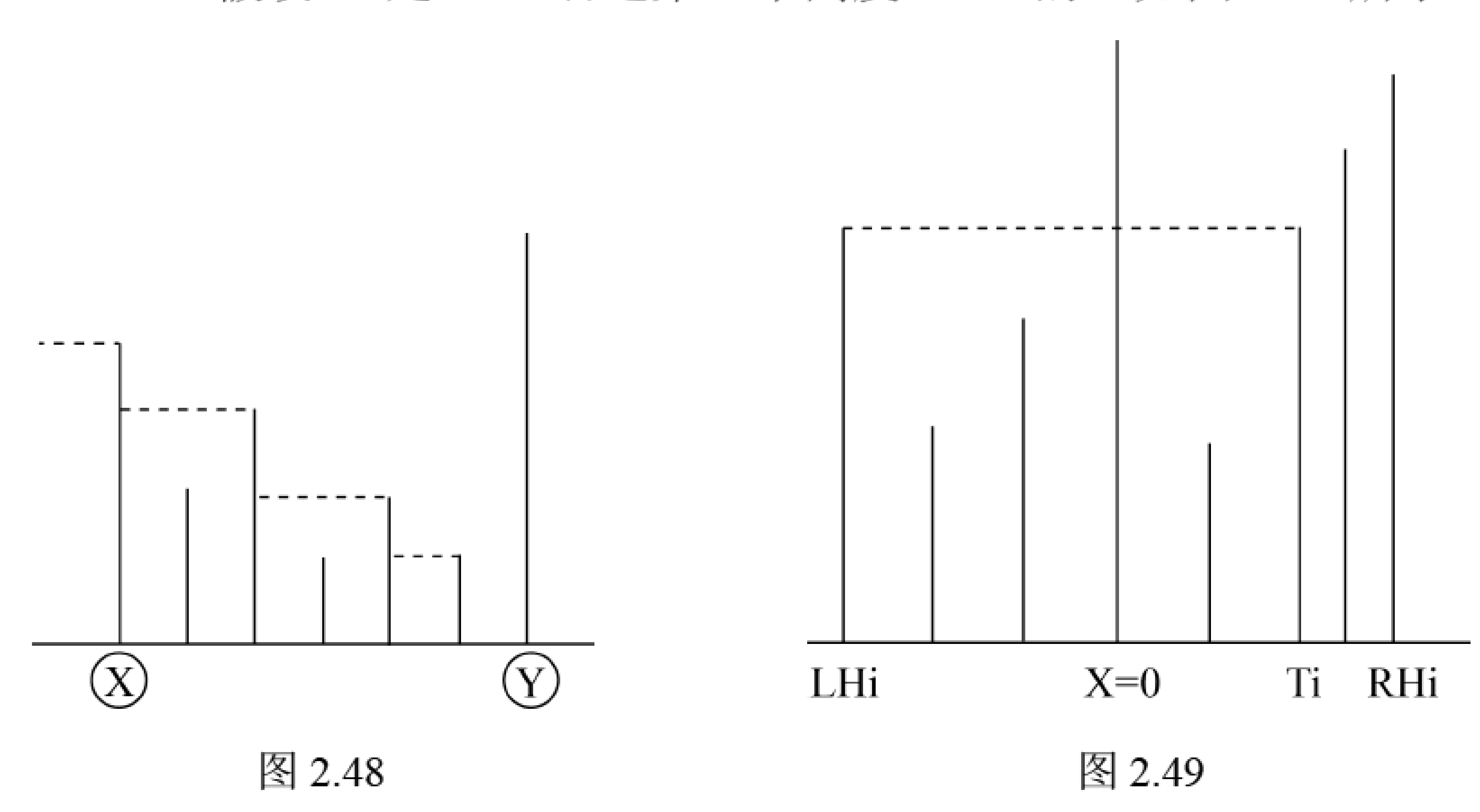


图 2.48　　　　图 2.49

那么水一定是首先接触到左边缘。而且水流分两部分，一部分从 Ti 流到 Ti 右边第一个高度大于 LH 的挡板的位置这里，另一部分从 LHi 向左溢出流到左边缘。假设前者所需水量为 rt，后者所需为 lt。

（1）如果 lt>rt，那么所求结果就是：LHi 和 Ti 之间高度为 LH 的矩形对应的水量也就是(LHi+Ti)*LH + lt + rt。

（2）如果 lt≤rt，rt 对应的部分只能灌满 lt 的水，左边就已经溢出了，所求结果就是：(LHi+Ti)*LH + 2*lt。

LH > RH 时的讨论可以类比以上情况。

为了简化处理，可以先假设两个挡板之间距离为 1，最后求出结果之后再乘以 2 输出即可。完整程序如下：

```
using namespace std;
const int MAXD = 1000+4;
```

```
vector<int> H;
int L, R, LHs[MAXD], RHs[MAXD], LH, RH, LHi, RHi;

int solve() {
   if(LH == RH) {
      int lt = 0, rt = 0;
      for(int i = L, h = LHs[L]; i > LHi; i--)
//从左边最高点溢出之后到边缘需要的水量
         lt += h, h = max(h, LHs[i-1]);
      for(int i = R, h = RHs[R]; i > RHi; i--)
//从右边最高点溢出之后到边缘需要的水量
         rt += h, h = max(h, RHs[i-1]);
      return (LHi + RHi + 1) * LH * 2 + min(lt, rt) * 2 * 2;
   }

   int T = min(LH, RH), LTi = 0, RTi = 0; //从左右两边第一个高度大于等于 T 的
   while(LTi < L && LHs[LTi] < T) LTi++;
   while(RTi < R && RHs[RTi] < T) RTi++;

   int lt = 0, rt = 0, t;
   if(LH < RH) {                                 //从左边溢出
      for(int i = L, h = LHs[L]; i > LHi; i--)
         lt += h, h = max(LHs[i-1], h);
      for(int i = RTi, h = T; RHs[i] <= T; i++)
         rt += h, h = max(RHs[i+1], h);
      t = lt > rt ? (lt+rt) : 2*lt;
   }
   if(LH > RH) {                                 //从右边溢出
      for(int i = R, h = RHs[R]; i > RHi; i--)
         rt += h, h = max(RHs[i-1], h);
      for(int i = LTi, h = T; LHs[i] <= T; i++)
         lt += h, h = max(h, LHs[i+1]);
      t = rt > lt ? (rt+lt) : 2*rt;
   }

   return t*2 + (RTi + LTi + 1) * T * 2;
}

int main() {
   int leftx, rightx;
   while(cin>>leftx>>rightx && leftx && rightx) {
      LH = RH = 0;
```



```
        L = (-leftx)/2, R = rightx/2;
        for(int i = leftx; i < 0; i += 2) {
            int xi = (-i)/2; cin>>LHs[xi];
            if(LH <= LHs[xi]) LH = LHs[xi], LHi = xi;
//左边离 0 最近的最高点，注意加上等号
        }

        for(int i = 1; i <= rightx; i += 2) {
            int xi = i/2; cin>>RHs[xi];
            if(RH < RHs[xi]) RH = RHs[xi], RHi = xi;
//右边离 0 最近的最高点及其位置
        }

        cout<<solve()<<endl;
    }
    return 0;
}
```

习题 8-25　有向图 D 和 E（From D to E and back, UVa11175）

对一个有 n 个结点的有向图 D，可以构造这样一个图 E，即 D 的每条边对应 E 的一个结点（例如，若 D 有一条边 uv，则 E 有个结点的名字叫 uv），对于 D 的两条边 uv 和 vw，E 中的两个结点 uv 和 vw 之间连一条有向边。E 中不包含其他边。

输入一个 m 个结点 k 条边的图 E（$0 \leqslant m \leqslant 300$），判断是否存在对应的图 D。E 中各个结点编号为 0～m-1。

提示：

> 虽然题目中 $m \leqslant 300$，实际上可以解决规模远超过这个限制的问题。

【分析】

在 E 图中，如果存在 i 和 j 结点到 x 都有边，而 i 和 j 中只有一个结点到 y 有边，则这个图 E 不可能转化来，因为在 D 中一条边不可能指向两个点（读者可以参考图 2.50 来思考）。因此暴力枚举 i、j 和 x 判断是否可行即可。

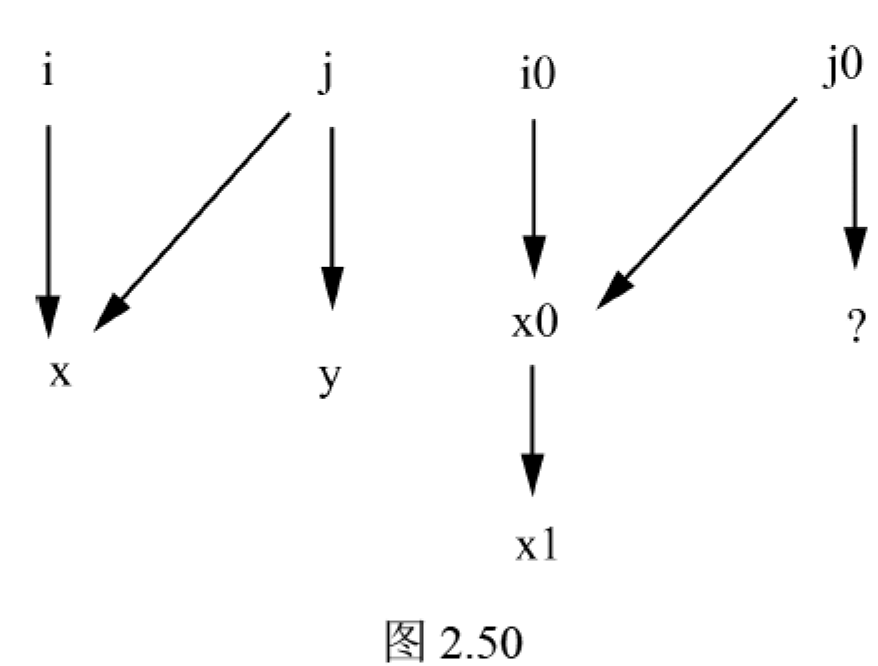


图 2.50


注意：

笔者无法构造上述做法的严格性证明，但是也无法找到反例[1] 。

完整程序（C++11）如下：

```
using namespace std;
const int maxm = 300 + 4;
vector<int> G[maxm], InvG[maxm];
int To[maxm];

bool check(int m) {
    _for(u, 0, m) {
        fill_n(To, m, 0);
        for(auto v : InvG[u])                //每一个由边出发到 u 的点
            for(auto x : G[v]) To[x]++;      //从 v 出发到达的每个结点计数加 1

        _for(v, 0, m) {
            if(To[v] == 0 || To[v] == InvG[u].size()) continue;
            return false;
        }
    }

    return true;
}

int main(){
    int N, m, k, x, y;
    scanf("%d", &N);
    for(int t = 1; t <= N; t++) {
        scanf("%d%d", &m, &k);
        _for(i, 0, m) G[i].clear(), InvG[i].clear();
        _for(i, 0, k) { scanf("%d%d", &x, &y); G[x].push_back(y); InvG[y].push_back(x); }
        bool valid = check(m);
        printf("Case #%d: %s\n", t, (valid ? "Yes" : "No"));
    }
    return 0;
}
```

习题 8-26　找黑圆（Finding [B]lack Circles, Rujia Liu’s Present 6, UVa12559）

输入一个 $h*w$ 的黑白图像（$30 \leqslant w,h \leqslant 100$），你的任务是找出图像中的圆。每个像素

[1] 如果有读者找到反例或者正确性证明，请联系笔者或出版社，我们会在重印时更正。

都是 1*1 的正方形，左上角像素的中心坐标为(0,0)，右下角像素的中心坐标为(w−1,h−1)。对于一个圆，它的圆周穿过（只是接触到像素边界不算）的像素都会被涂黑（用 1 表示）。没有被任何圆穿过的像素仍然是白色（用 0 表示）。圆心保证在整点处，半径保证是 1～5 的整数。最多有 2%的黑点会变成白点。

提示：

方法有多种，尽情发挥创造力吧。

【分析】

因为数据量不是特别大，所有可能的圆的个数上限是 50*30*100，可以考虑进行暴力搜索每一种可能的半径以及坐标(r,x,y)。

对于(r,x,y)来说，要循环判断圆上的每个点，因为边上的点最多也就是 100 个。可以遍历 100 个圆心角角度对应的点来判断，为了提高速度，可以采用如下技巧：

（1）随机取 100 个角度，看看这个角度对应的边上的点是不是存在。

（2）如果已经判断超过 10 个且有一半不符合，说明一定不存在对应的圆，直接返回即可。

完整程序如下：

```
using namespace std;
int readint() { int x; cin>>x; return x; }
const double pi=acos(-1);

struct circle { //表示一个圆的结构
   int r, x, y;
   circle(int _r, int _x, int _y = 0) : r(_r), x(_x), y(_y) { }
   bool operator<(const circle& c) const {
      if(r != c.r) return r < c.r;
      if(x != c.x) return x < c.x;
      return y < c.y;
   }
};

ostream& operator<<(ostream& os, const circle& c) {
   char buf[128];
   sprintf(buf, " (%d,%d,%d)", c.r, c.x, c.y);
   return os<<buf;
}

vector<string> lines;
int w, h;
bool inRange(int x, int left, int right) {
   if(left > right) return inRange(x, right, left);
```

```
    return left <= x && x <= right;
}

int main()
{
    int T = readint();
    vector<circle> ans;
    _for(t, 1, T+1) {
        ans.clear();
        cin>>w>>h;
        lines.resize(h);
        _for(j, 0, h) cin>>lines[j];

        _for(r, 5, 51) _for(x, r, w-r+1)
            _for(y, r, h-r+1) { //r是坐标, x 是列, y是行
            int all = 0, per = 0;
            _for(i, 0, 100){
                double th = rand()/(RAND_MAX+1.0) * 2 * pi; //随机选取一个角度
                int cx = (int)(x+r*cos(th)+0.5), cy = (int)(y+r*sin(th)+0.5);
                if(inRange(cx, 0, w-1) && inRange(cy, 0, h-1)
                    && lines[cy][cx]=='1') per++;
                all++;
                if(all > 10 && 2*per < all) break;          //剪枝
            }

            if(per / (double)all > 0.8) ans.push_back(circle(r, x, y));
        }

        cout<<"Case "<<t<<": "<<ans.size();
        _for(i, 0, ans.size()) cout<<ans[i];
        cout<<endl;
    }

    return 0;
}
```

2.7 动态规划初步

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 9 章。

习题 9-1 最长的滑雪路径（Longest Run on a Snowboard, UVa 10285）

在一个 $R*C$（$R,C\leqslant 100$）的整数矩阵上找一条高度严格递减的最长路。起点任意，但

每次只能沿着上下左右 4 个方向之一走一格，并且不能走出矩阵外。如图 2.51 所示，最长路就是按照高度 25、24、23、…、2、1 这样走，长度为 25。矩阵中的数均为 0～100。

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

图 2.51

【分析】

用 D[i,j]来表示从[i,j]开始能走的高度严格递减的最长路，则很容易发现 D[i,j]的状态转移方程：D[i,j] = max(1 + D[i1,j1])，其中(i1,j1)是比[i,j]高度小的 4 个相邻的格子之一。边界条件是当没有符合条件的 i1 和 j1 时，D[i,j]=1。使用记忆化搜索即可，时间复杂度是 $O(R*C)$。

习题 9-2　免费糖果（Free Candies, UVa10118）

桌上有 4 堆糖果，每堆有 N（$N \leqslant 40$）颗。佳佳有一个最多可以装 5 颗糖的小篮子。他每次选择一堆糖果，把最顶上的一颗拿到篮子里。如果篮子里有两颗颜色相同的糖果，佳佳就把它们从篮子里拿出来放到自己的口袋里。如果篮子满了而里面又没有相同颜色的糖果，游戏结束，口袋里的糖果就归他了。当然，如果佳佳足够聪明，他有可能把堆里的所有糖果都拿走。为了拿到尽量多的糖果，佳佳该怎么做呢？

【分析】

初步看，状态包含 4 堆糖果的高度 hs[4]以及当前篮子内是否有各色糖果。如果全部考虑进去，空间复杂度会大到无法接受。思考之后会发现，只要有两个相同的糖果就会被拿出来，对于特定的 hs，篮子中的状态就可以确定。

可以对 hs[4]进行回溯搜索，每一步尝试从一堆糖果顶端取出一个糖果。同时记忆 hs 对应的状态，减少重复计算。算法的时间和空间复杂度都刚好是 $O(n^4)$。完整程序如下：

```
using namespace std;

const int MAXH = 40+4, COLOR = 20;
int n, DP[MAXH][MAXH][MAXH][MAXH], pile[4][MAXH];

struct Basket {
   int color[COLOR + 4], size;
   Basket() : size(0){ memset(color, 0, sizeof(color)); }
   bool isFull() { return size == 5; }
   void take(int c) { //取出颜色为 c 的糖果
      assert(color[c]);
      color[c] = 0;
      size--;
   }
   void put(int c) { //放入颜色为 c 的糖果
      assert(!isFull());
      assert(!color[c]);
      color[c] = 1;
      size++;
```

```
    }
};
//给定篮子的状态和每个pile的高度，返回最多能取出多少个糖果
int dfs(Basket& bkt, vector<int>& hs) {
    int& ans = DP[hs[0]][hs[1]][hs[2]][hs[3]];
    if(ans != -1) return ans;
    ans = 0;
    if(bkt.isFull()) return ans;

    _for(i, 0, hs.size()){
        int& h = hs[i];
        if(h <= 0) continue;
        int sum = 0, top = pile[i][h-1];
        h--; //尝试把pile[i]顶端的糖果取出来
        if(bkt.color[top]) {
            bkt.take(top);
            sum = dfs(bkt, hs) + 1;
            bkt.put(top);
        }
        else if(!bkt.isFull()){
            bkt.put(top);
            sum = dfs(bkt, hs);
            bkt.take(top);
        }
        ans = max(ans, sum);
        h++;
    }

    return ans;
}
int main(){
    vector<int> hs(4);
    while(cin>>n && n){
        hs.assign(4, n);
        memset(DP, -1, sizeof(DP));
        _for(j, 0, n) _for(i, 0, 4) cin>>pile[i][n-j-1];
        Basket bkt;
        cout<<dfs(bkt, hs)<<endl;
    }
}
```

习题 9-3　切蛋糕（Cake Slicing, ACM/ICPC Nanjing 2007, UVa1629）

有一个 n 行 m 列（$1\leqslant n,m\leqslant 20$）的网格蛋糕上有一些樱桃。每次可以用一刀沿着网格

线把蛋糕切成两块，并且只能够直切不能拐弯。要求最后每一块蛋糕上恰好有一个樱桃，且切割线总长度最小。如图 2.52 所示是一种切割方法。

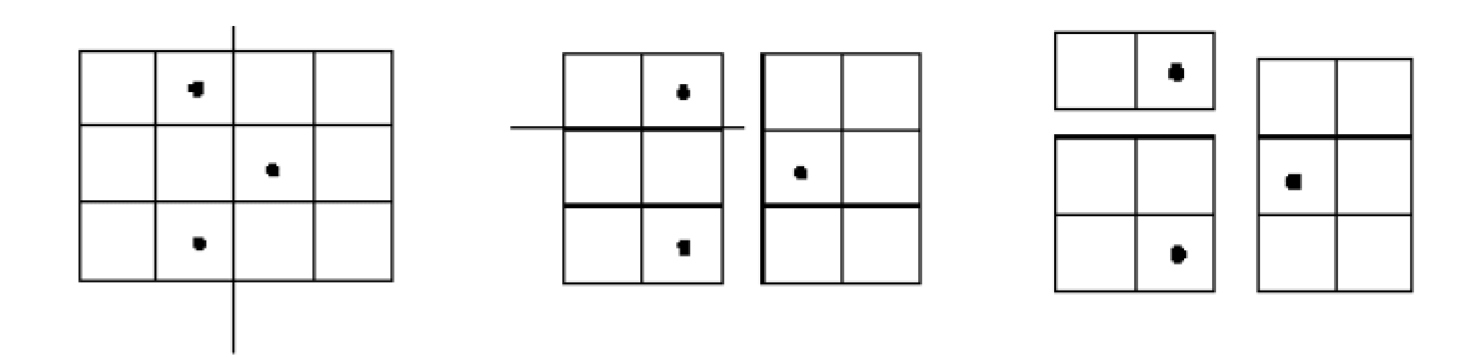


图 2.52

【分析】

有一个明显的递归子结构，就是包含 1 个以上樱桃的子矩形 r、c、w、h。左上角$[r,c]$，宽为 w、高为 h。

记 $D(r,c,w,h)$（下文简称 d）为这个矩形的最小切割线长度，对于这个矩形来说，只有横竖两种切法：

（1）遍历所有合法的横切，$d = \min(d, \min(w+D(r+i, c, w, h-i)+D(r, c, w, i))), i \in [1,h]$。

（2）遍历所有合法的竖切，$d = \min(d, \min(h+D(r,c,i,h)+D(r, c+i, w-i, h))), i \in [1,w]$。

边界条件是(r,c,w,h)对应的子矩形中刚好只有 1 个樱桃，此时 d=0。遍历时需要保证切开的两个矩形都至少包含 1 个樱桃。

在遍历过程中需要求出每个子矩形的樱桃个数，可以用求 D 类似的递归结构来记忆化搜索所有子矩形的樱桃个数。算法的时空复杂度都是 $O(n^3m^3)$。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 20 + 1;
int n, m, k, D[MAXN][MAXN][MAXN][MAXN], Chery[MAXN][MAXN][MAXN][MAXN];

void Check(int r, int c, int width, int height) {
   assert(width >= 1);
   assert(height >= 1);
   assert(0 <= r && r < n);
   assert(0 <= c && c < m);
   assert(r + height <= n);
   assert(c + width <= m);
}

//区域[r,c,width,height]内的樱桃个数
int CR_Cnt(int r, int c, int width, int height){
   int& ans = Chery[r][c][width][height];
   if(ans != -1) return ans;

   if(width > 1)
      return ans = CR_Cnt(r, c, 1, height) + CR_Cnt(r, c+1, width-1, height);
```

```
    return ans = CR_Cnt(r, c, 1, 1) + CR_Cnt(r+1, c, 1, height-1);
}

int dp(int r, int c, const int W, const int H) {
    int &ans = D[r][c][W][H];
    if(ans != -1) return ans;
    if(CR_Cnt(r,c,W,H) == 1) return ans = 0;

    auto updateAns = [&](int a){
        if(ans == -1) ans = a;
        else ans = min(ans, a);
    };
    //遍历横着切
    _for(h, 1, H) //h上半部分的高度
        if(CR_Cnt(r,c,W,h) >= 1 && CR_Cnt(r+h,c,W, H-h) >= 1)
            updateAns(W + dp(r+h, c, W, H-h) + dp(r, c, W, h));

    //遍历竖着切
    _for(w, 1, W) //w左边的宽度
        if(CR_Cnt(r,c,w,H) >= 1 && CR_Cnt(r,c+w,W-w,H) >= 1)
            updateAns(H + dp(r,c,w,H) + dp(r,c+w,W-w,H));

    return ans;
}

int main(){
    for(int kase = 1,r,c; scanf("%d%d%d", &n, &m, &k) == 3; kase++){
        memset(D, -1, sizeof(D));
        memset(Chery, -1, sizeof(D));
        _for(i, 0, n) _for(j, 0, m) Chery[i][j][1][1] = 0;
        _for(i, 0, k) {
            scanf("%d%d", &r, &c);
            Chery[r-1][c-1][1][1] = 1;
        }

        printf("Case %d: %d\n", kase, dp(0,0,m,n));
    }
    return 0;
}
```

习题 9-4　串折叠（Folding, ACM/ICPC NEERC 2001, UVa1630）

给一个由大写字母组成的长度为 n（$1 \leqslant n \leqslant 100$）的串，“折叠”成一个尽量短的串。

例如 AAAAAAAAAABABABCCD 折叠成 9(A)3(AB)CCD。折叠是可以嵌套的，例如 NEERCYESYESYESNEERCYESYESYES 可以折叠成 2(NEERC3(YES))。多解时可以输出任意解。

【分析】

参考“最优矩阵链乘”问题的思路，进行区间规划。记输入串为 S，DP(L,R)表示这个区间的最优折叠结果的字符串表示，用 STL 的 string 来存储。则状态转移方程如下：

（1）边界条件：L = R 时，DP(L,R) ="S[L]"。

（2）L<R，则考虑两种策略，取长度最短的方案：

- ❑ 把区间切分成两部分，分别折叠然后拼接，遍历所有的区间切分方案[L,k]和[k+1,R]（L≤k<R），求出令 DP(L,k)和 DP(k+1,R)长度之和最小的 k 值 kMin，则有 DP(L,R) = DP(L, kMin)+DP(kMin+1,R)。
- ❑ 如果 S[L,R]是周期串，首先求出最小的重复串长度 cycle(0<cycle≤(L−R+1)/2)，记 rep=(L−R+1)/cycle，也就是重复次数。这样把区间变成 DP(L,L+cycle−1)的 rep 次折叠。DP(L,R) ="cnt("+"DP[L][L+cycle-1]"+")"。例如，“ABABABAB”变成“4(AB)”。

所求结果就是 DP(0,n−1)，算法的时间空间复杂度均为 $O(n^3)$。完整程序如下：

```
using namespace std;
const int INF = 0x3f3f3f3f, MAXN = 104;
string S, Fold[MAXN][MAXN];

int getMinCycle(int l, int r){
   int segLen = r-l+1;
   _rep(cycle, 1, segLen/2){
      if(segLen%cycle) continue;
      bool flag = true;
      _rep(j, l, r-cycle){
         if(S[j] == S[j+cycle]) continue;
         flag = false;
         break;
      }
      if(flag) return cycle;
   }
   return 0;
}

string& solve(int l, int r){
   string& ans = Fold[l][r];
   if(!ans.empty()) return ans;
   if(l == r) return ans = S[l];

   int minK, ansLen = INF;
```

```
    _for(i, l, r){
        int len = solve(l,i).size() + solve(i+1, r).size();
        if(len < ansLen) minK = i, ansLen = len;
    }
    ans += solve(l, minK);
    ans += solve(minK+1, r);

    int cycle = getMinCycle(l, r);
    if(cycle){
        stringstream ss;
        ss<<(r-l+1)/cycle<<"("<<solve(l, l+cycle-1)<<")";
        if(ss.tellp() < ans.size()) ss>>ans;
    }
    return ans;
}

int main(){
    while(cin>>S){
        int n = S.size();
        _for(i, 0, n) _for(j, 0, n) Fold[i][j].clear();
        cout<<solve(0, n-1)<<endl;
    }
}
```

习题 9-5　邮票和信封（Stamps and Envelope Size, ACM/ICPC World Finals 1995, UVa242）

假定一张信封最多贴 5 张邮票，如果只能贴 1 分和 3 分的邮票，可以组成面值 1～13 以及 15，但不能组成面值 14。我们说：对于邮票组合{1,3}以及数量上限 S=5，最大连续邮资为 13。1～13 和 15 的组成方法如表 2.1 所示。

表 2.1

1=1	2=1+1	3=3	4=1+3	5=1+1+3
6=3+3	7=1+3+3	8=1+1+3+3	9=3+3+3	10=1+3+3+3
11=1+1+3+3+3	12=3+3+3+3	13=1+3+3+3+3	14 无法表示	15=3+3+3+3+3

输入 S（$S \leqslant 10$）和若干邮票组合（邮票面值不超过 100），选出最大连续邮资最大的一个组合。如果有多个并列，邮票组合中邮票的张数应最多。如果还有并列，邮票从大到小排序后字典序应最小。

【分析】

对于邮票组合 C，令 DP[i]表示邮资 i 至少需要多少张来自于 C 的邮票才能组合起来。则 DP[i] = min(DP[i−x]+1, x∈C 且 x≤i)。则对于 C 来说最大连续邮资就是第一个符合

DP[i+1]>S 的 i。这样就可以求出每个组合的最大连续邮资。时间和空间复杂度都为 *O*(100*maxS)。完整程序（C++11）如下：

```
using namespace std;
int readint() { int x; scanf("%d", &x); return x; }

const int MAXN = 10 + 1, MAXS = MAXN, MAXC = MAXS * 100 + 100 + 4;
int S, N, DP[MAXC];

struct StampSet {
   vector<int> D;
   int maxCover;
   void output(){ for(auto e : D) printf("%3d", e); }
   StampSet& input(){
      D.clear(); maxCover = 0;
      int n = readint();
      while(n--) D.push_back(readint());
      sort(D.begin(), D.end());
      return *this;
   }
   void getMaxCover() {
      int i = 0;
      fill_n(DP, MAXC, 0);
      DP[i] = 0;
      while(true) {
         i++;
         int ans = INT_MAX;
         for(int j = 0; j < D.size() && D[j] <= i; j++)
            ans = min(ans, DP[i-D[j]] + 1);
         if(ans > S) break;
         else DP[i] = ans;
      }
      maxCover = i - 1;
   }

   bool operator<(const StampSet& rhs) const {
      if(maxCover != rhs.maxCover) return maxCover > rhs.maxCover;
      if(D.size() != rhs.D.size()) return D.size() < rhs.D.size();
      for(int i = D.size()-1; i >= 0; i--)
         if(D[i] != rhs.D[i]) return D[i] < rhs.D[i];
      return true;
   }
};
```

```
StampSet C[MAXN];
int main() {
   while(scanf("%d", &S) && S) {
      N = readint();
      _for(i, 0, N) C[i].input().getMaxCover();
      auto mss = min_element(C, C+N);
      printf("max coverage = %3d :", mss->maxCover);
      mss->output();
      puts("");
   }
   return 0;
}
```

习题 9-6　电子人的基因（Cyborg Genes, UVa10723）

输入两个 A～Z 组成的字符串（长度均不超过 30），找一个最短的串，使得输入的两个串均是它的子序列（不一定连续出现）。你的程序还应统计长度最短的串的个数。如 ABAAXGF 和 AABXFGA 的最优解之一为 AABAAXGFGA，一共有 9 个解。

【分析】

参考 LCS 问题的思路。记输入的两个字符串为 S1 和 S2，定义 $pa(i,j)$为 $S1[1\cdots i]$和 $S2[1\cdots j]$的公共父串的最短长度。则 pa 的状态转移方程如下：

（1）$pa(i,j) = \min(pa(i-1, j) + 1, pa(i,j-1) + 1)$，其中 $S1[i]\neq S2[j]$，对应父串的最后一位是使用 $S1[i]$还是 $S2[j]$。

（2）$pa(i,j) = pa(i-1,j-1) + 1$，其中 $S1[i]=S2[j]$，则父串的最后一位是确定的。

（3）边界条件是：当 i=0 或者 j=0 时，$pa(i,j) = \max(i, j)$，则父串一定是 $S1[1\cdots i]$和 $S2[1\cdots j]$其中之一。

然后求最短父串的方案个数：定义 $pac(i,j)$为 $S1[1\cdots i]$ and $S2[1\cdots j]$的最短公共父串的个数。$S1[i] = S2[j]$ 时，$pac(i,j) = pac(i-1,j-1)$，因为父串的最后一位只有 1 种选择。否则记 $p1 = pa(i-1, j)$，$p2 = pa(i, j-1)$ ，则有：

- $pac(i,j) = pac(i-1, j) + pac(i, j-1)$，此时 p1=p2，表示父串最后一位可以使用 $S1[i]$和 $S2[j]$两种方案。
- $pac(i,j) = pac(i-1, j)$, $p1 < p2$，父串必须使用 $S1[i]$才能保证最短。
- $pac(i,j) = pac(i, j-1)$, $p1 > p2$，父串必须使用 $S2[j]$才能保证最短。
- 边界条件是：当 i=0 或者 j=0 时，$pac(i, j)$=1，父串只有 1 种选择。

时间复杂度和空间复杂度都为 $O(|S1|*|S2|)$。注意，输入的 S1 和 S2 长度可能是 0，输入时要用 gets 或者 STL 里面的 getline 而不能用 scanf。完整程序如下：

```
using namespace std;
typedef long long LL;
```



```
const int MAXL = 32;
char S1[MAXL], S2[MAXL];
LL len1, len2, Pa[MAXL][MAXL], Pac[MAXL][MAXL];

//求 SL[i1, i2] : S1[1...i1] 和 S2[1...i2]的公共父串的最短长度
LL dpPa(int i1, int i2) {
    LL& ans = Pa[i1][i2];
    if(ans != -1) return ans;
    if(i1 == 0 || i2 == 0) return ans = max(i1,i2);
    if(S1[i1] == S2[i2]) return ans = dpPa(i1-1, i2-1) + 1;
    return ans = min(dpPa(i1-1,i2), dpPa(i1,i2-1)) + 1;
}

//长度为 S1[1...i1] S2[1...i2]的最短长度公共父串的个数
LL dpPac(int i1, int i2) {
    LL& ans = Pac[i1][i2];
    if(ans != -1) return ans;
    if(i1 == 0 || i2 == 0) return ans = 1;
    if(S1[i1] == S2[i2]) return ans = dpPac(i1-1, i2-1);

    LL sl1 = dpPa(i1-1, i2), sl2 = dpPa(i1, i2-1);
    if(sl1 == sl2)  ans = dpPac(i1-1, i2) + dpPac(i1, i2-1);
    else if(sl1 < sl2)  ans = dpPac(i1-1, i2);
    else ans = dpPac(i1, i2-1);
    return ans;
}

int main(){
    int T; scanf("%d\n", &T);
    _rep(t, 1, T){
        gets(S1+1), gets(S2+1);
        len1 = strlen(S1+1), len2 = strlen(S2+1);
        memset(Pa, -1, sizeof(Pa)), memset(Pac, -1, sizeof(Pac));
        printf("Case #%d: %lld %lld\n", t, dpPa(len1, len2), dpPac(len1, len2));
    }
}
```

习题 9-8　阿里巴巴（Alibaba, ACM/ICPC SEERC 2004, UVa1632）

直线上有 n(n≤10000)个点，其中第 i 个点的坐标是 x_i，且它会在 d_i 秒之后消失。Alibaba 可以从任意位置出发，求访问完所有点的最短时间。无解输出 No solution。

【分析】

最优的访问策略是在任意时刻，访问过的点要形成一个连续区间，中间不存在未访问过的点（顺路都可以把那个点访问了）。

定义：

- D(i,j,0)：访问 i 到 j，最后停在 i 点所需要的最少时间。
- D(i,j,1)：访问 i 到 j，最后停在 j 点所需要的最少时间。

则状态转移方程如下：

- $D(i,j,0) = \min(D(i+1, j,0) + x_{i+1} - x_i, D(i+1, j,1)+ x_j - x_i)$，其中有两种策略：先访问 i+1 到 j，停在 i+1 再去 i，或者停在 j 再去 i。
- $D(i, j,1) = \min(D(i, j-1,1)+x_j - x_{j-1}, D(i, j-1,0)+ x_j - x_i)$，其中有两种策略：先拿 i 到 j−1 的物品，停在 j−1 再去拿 j，或者停在 i 再去拿 j。

时间和空间复杂度都为 $O(2*n^2)$。需要注意的是，题目没有明确说，但是可以假设 Alibaba 单位时间移动一个单位的距离。

习题 9-9　仓库守卫（Storage Keepers, UVa10163）

你有 n（n≤100）个相同的仓库。有 m（m≤30）个人应聘守卫，第 i 个应聘者的能力值为 P_i（1≤P_i≤1000）。每个仓库只能有一个守卫，但一个守卫可以看守多个仓库。如果应聘者 i 看守 k 个仓库，则每个仓库的安全系数为 P_i/K 的整数部分。没人看守的仓库安全系数为 0。

你的任务是招聘一些守卫，使得所有仓库的最小安全系数最大，在此前提下守卫的能力值总和（这个值等于你所需支付的工资总和）应最小。

【分析】

类似于背包问题，令 F(i,j)表示前 i 个人，管理 j 个仓库的最小安全系数最大值，则有：

- F(i,0) = INF，0 个仓库最小安全系数可以认为是 INF，方便递推。
- F(1,j) = P_1/j，每个仓库的安全系数都是 P_1/j。
- 记 k 为第 i 个人管理的仓库个数（0≤k≤j），G(k)为第 i 个人管理 k 个仓库时，前 j 个仓库最小安全系数的最大值。则 k=0 时，G(0)=F(i−1, j)，否则 $G(k) = \min(F(i-1, j-k), P_i/k)$。$F(i,j)=\max(G(k))$。

记 mx = F(m,n)，然后就是求工资总和：G(i,j)表示前 i 个人，管理 j 个仓库的达到最大安全系数前提下，这 i 个人能力总和的最小值。

- G(i,0) = 0，0 个仓库只需要 0 个人管。
- G(1,j) = P_i/j≥mx ? P_i : INF，只能选择让安全系数超过最大值的人来管理这 G 个仓库。
- $G(i,j) = \min(G(i-1, j-k)+P_i)$，其中 P_i/k≥mx && 0≤k≤j，这里依然是对第 i 个人管理的仓库个数 k 进行决策。

需要注意的是，上述状态转移过程中，如果 k = 0，则有 P_i/k = INF。空间复杂度为 $O(n*m)$，时间复杂度为 $O(n^2*m)$。完整程序（C++11）如下：

```
using namespace std;
```



```
const int MAXN = 128, MAXM = 32, INF = 0x3f3f3f3f;
int N, M, mx, P[MAXM], F[MAXM][MAXN], G[MAXM][MAXN];
//F[i][j] 表示前 i 个工人要管理 j 个仓库能达到的最高安全度
int dpF(int i, int j) {
    int& d = F[i][j];
    assert(i >= 1);
    if (d >= 0) return d;
    if (j == 0) return d = INF;
    if (i == 1) return d = (j == 0) ? INF : (P[i] / j);

    d = dpF(i - 1, j); // k = 0
    _rep(k, 1, j) d = max(d, min(dpF(i - 1, j - k), P[i]/k));
    return d;
}

//g[i][j]表示前 i 个人，管理 j 个仓库达到最大安全度所需要的最小价钱
int dpG(int i, int j) {
   int& d = G[i][j];
   assert(i>=1);
   if(d >= 0) return d;
   if(j == 0) return d = 0;
   if(i == 1) {
      if(P[i]/j >= mx) return d = P[i];
      return d = INF;
   }
   d = dpG(i-1, j);
   _rep(k, 1, j) if(P[i]/k >= mx) d = min(d, dpG(i-1,j-k)+P[i]);

    return d;
}

int main() {
    while (scanf("%d%d", &N, &M) == 2 && N) {
        memset(F, -1, sizeof(F)), memset(G, -1, sizeof(G));
        _rep(i, 1, M) scanf("%d", &(P[i]));
        if ((mx = dpF(M,N)) == 0) { puts("0 0"); continue; }
        printf("%d %d\n", mx, dpG(M, N));
    }
    return 0;
}
```

习题 9-10　照亮体育馆（Barisal Stadium, UVa10641）

输入一个凸 n（$3 \leqslant n \leqslant 30$）边形体育馆和多边形外的 m（$1 \leqslant m \leqslant 1000$）个点光源，每个点光源都有一个费用值。选择一组点光源，照亮整个多边形，使得费用值总和尽量小。如图 2.53 所示，多边形 ABCDEF 可以被两组光源{1,2,3}和{4,5,6}照亮。光源的费用决定了哪组解更优。

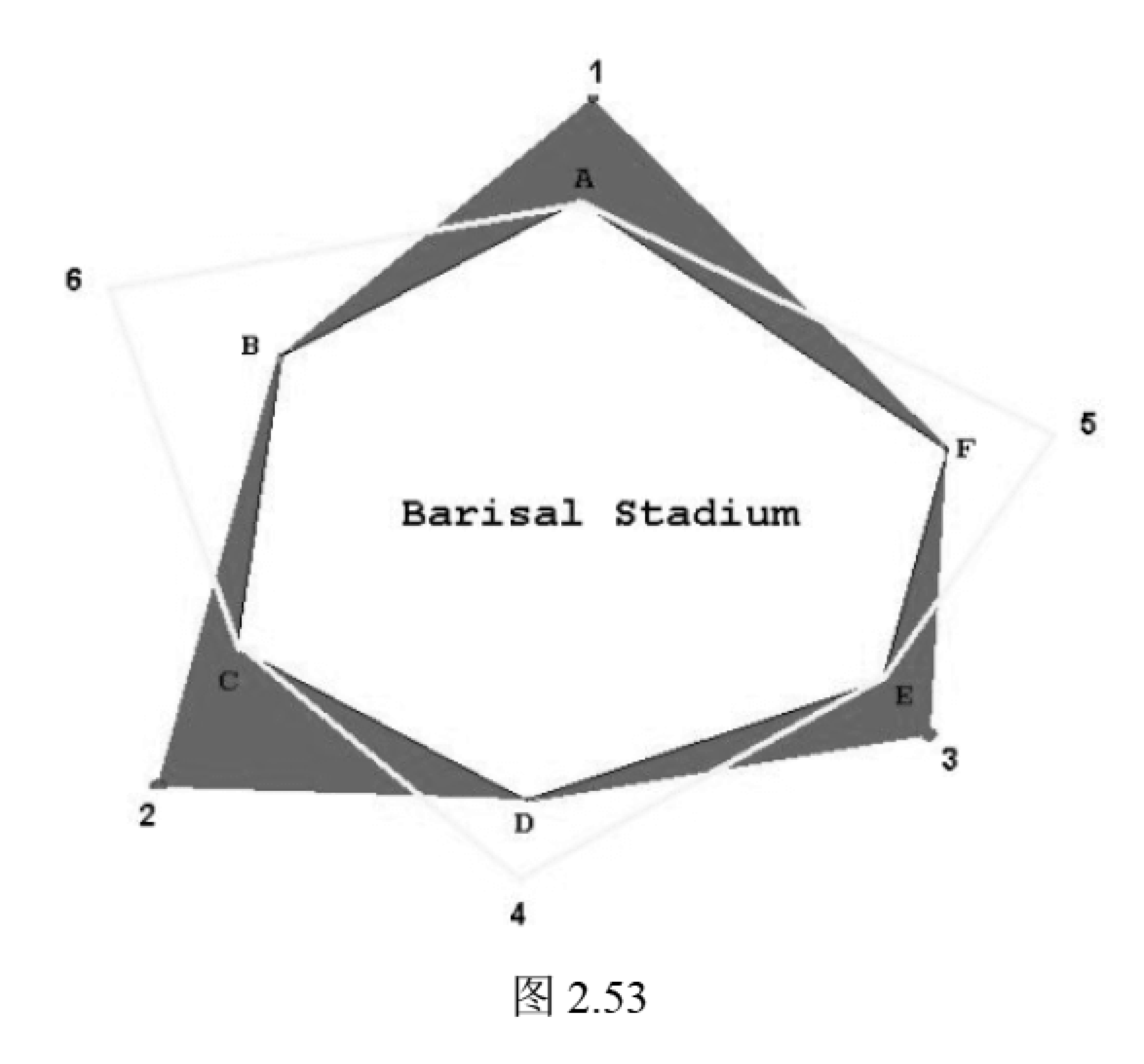


图 2.53

【分析】

照亮整个多边形，也就是要选择一组费用最小的光源，使得每个顶点都被照亮。首先需要计算出每个光源可以照到哪些顶点。例如，图 2.53 中光源 1 可以照亮边 AB（包含 A、B 两个顶点），一定存在多边形内部的一个点，刚好和点 1 分别位于 AB 的不同侧。进一步，先通过把所有顶点坐标求平均值得到一个多边形内部的点 O，再使用叉积来计算每个光源可以照亮的边和顶点。

而本题中所有顶点形成一个环，不难想到首先要把环变成直线，可以把区间[0,n)扩大两倍成为[0,2n)，其中 $i \geqslant n$ 对应原来的点 $i-n$，然后再预处理出每个光源能够照亮的顶点编号区间[L,R]。一组光源只要能把[0,2n)的任意一个包含 n 个顶点的子区间内的所有顶点都照亮，就是一组符合要求的解。现在就是要求出所有解中费用最小的。

对于顶点 i（$0 \leqslant i < n$）：定义 D(j)为顶点编号区间[i, j)内的顶点都被照射到所需的最小费用，则本题的解就是 min(D($i+n$)), $0 \leqslant i < n$。对于每个 i，从小到大遍历顶点编号 j（$i \leqslant j < i+n$），然后考虑每个能照射到 j 的光源 lt，记 r = min(lt.R,$i+n$)，表示使用了 lt 之后，能够照射到的右边界。分是否使用 lt 两种情况考虑进行状态转移，用 D(j)更新 D(r)：D(r) = min(D(r), D(j) + lt.c)，其中 lt.c 表示光源 lt 的费用。

时间复杂度为 $O(2n^2m)$，空间复杂度为 $O(n)$。完整程序如下：

```
using namespace std;
const double eps = 1e-7, Pi = acos(-1);
double dcmp(double x) { if(fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
```

```
bool dcmp(double x, double y) { return dcmp(x-y) < 0; }

double operator*(const Vector& A, const Vector& B) { return A.x*B.x +
A.y*B.y; }
double Length(const Vector& A) { return sqrt(A*A); }
double Cross(const Vector& A, const Vector& B) { return A.x*B.y - A.y*B.x; }

istream& operator>>(istream& is, Point& p) { return is>>p.x>>p.y; }

//灯的照射范围为[l, r)，代价为c，如果r>N，那么表示r-N号顶点
struct Light{ int l, r, c; };

const int MAXN = 32, MAXM = 1000+4, INF = 0x3f3f3f3f;
int N, M;
Point V[MAXN], O;
Light lights[MAXM];

//lt能照到线段[a,b]吗
bool canCover(const Point& lt, const Point& a, const Point& b) {
   return dcmp(Cross(lt-a, b-a) * Cross(O-a, b-a)) < 0;
}
//坐标为p的灯lt能照亮顶点编号区间吗，预处理出来
void getCover(Light& lt, const Point& p) {
   vector<bool> flag(N);
   _for(i, 0, N) flag[i] = canCover(p, V[i], V[i+1]);

   if(flag[0] && flag[N-1]) {
      int l = N - 1, r = N;
      while(flag[l]) lt.l = l--;
      while(flag[r-N]) lt.r = r++;
   } else {
      int l = 0, r = N-1;
      while(l<N && !flag[l]) l++;
      lt.l = l;
      while(r>=0 && !flag[r]) r--;
      lt.r = r;
   }

   lt.r++;
   if(lt.r < lt.l) lt.r += N;
}
```

```
int solve() {
    int ans = INF;
    vector<int> D(2*N);
    _for(i, 0, N) { //从第 i 个顶点开始考虑
        fill(D.begin(), D.end(), INF);
        D[i] = 0;    //D[j]表示 i 到第 j 个顶点之间的边都被照射到的最小代价
        _for(j, i, i+N) _for(k, 0, M){       //l 为区间长度，遍历每个灯
            const Light& lt = lights[k];
            if(lt.l > j) continue;            //lt 照不到 j
            int r = min(lt.r, i+N);           //使用了灯 lt 之后能照到的右边界
            D[r] = min(D[r], D[j] + lt.c);    //是否使用灯 lt
        }
        ans = min(ans, D[i+N]);
    }
    return ans;
}

int main() {
    while(cin>>N && N) {
        O.x = 0, O.y = 0;
        _for(i, 0, N) cin>>V[i], O += V[i];
        O.x /= N, O.y /= N;
        V[N] = V[0];

        cin>>M;
        Point p;
        _for(i, 0, M) {
            cin>>p>>lights[i].c;
            getCover(lights[i], p);
        }

        int ans = solve();
        if(ans == INF) cout<<"Impossible."<<endl;
        else cout<<ans<<endl;
    }
    return 0;
}
```

习题 9-11　禁止的回文子串（Dyslexic Gollum, ACM/ICPC Amritapuri 2012, UVa1633）

输入正整数 n 和 k（$1 \leqslant n \leqslant 400$，$1 \leqslant k \leqslant 10$），求长度为 n 的 01 串中有多少个不含长度至少为 k 的回文连续子串。例如 $n=k=3$ 时只有 4 个串满足条件：001, 011, 100, 110。

【分析】

首先，长度为 a+2 的回文，去掉两端字符之后一定是长度为 a 的回文。也就是说，只要保证不包含长度为 k 和 k+1 的回文串，则一定不包含更长的回文串。

题目中 k 比较小（$k\leqslant 10$），可以把长度为 k 的串作为一个整体，使用一个整数来记录（≤1024），这样可以直接使用位运算。否则用字符串记录，还需要做时间复杂度高得多的字符串运算。

令 F(i,b)表示已经确定了从左到右的前 i 位，其中最右边 k 位对应的整数为 b，剩下的 $n-i$ 位上所有方案的个数。

首先引入以下变量：

（1）$b_0 = (b<<1)$，表示 b 左移一位，然后右边补 0，得到的 k+1 位串。

（2）$b_1 = (b<<1)+1$，表示 b 左移一位，然后右边补 1，得到的 k+1 位串。

（3）$c_0= b_0\& ((1<<k)-1)$，表示取 b_0 最右边 k 位得到的 k 位串。

（4）$c_1= b_1\& ((1<<k)-1)$，表示取 b_1 最右边 k 位得到的 k 位串。

c_0 和 c_1 分别表示确定了第 i+1 位之后的最右边 k 位串，分别对应第 i+1 位为 0 和 1 两种情况，则状态转移方程如下：$F(i, b) = F(i+1, c_0) + F(i+1,c_1)$。上述方程中，要求 b_0、b_1、c_0、c_1 不是回文，并且其最右边 k 位也不是回文。

边界条件是当 $i = n$ 时 F=1。则最终答案就是$\sum F(k, b)$, 其中 b 是所有 k 位的非回文。算法的时间和空间复杂度均为 $O(n*2^k)$。

☞注意：

（1）可以提前用 DFS 把 k 位和 k+1 位的回文搜索出来保存用来判断 b_0 和 b_1 是否是回文串。
（2）当 $k>n$ 时，答案为 2^n，因为任意串都满足要求。

完整程序如下：

```
using namespace std;
typedef long long LL;
const LL MOD = 1000000007;
const int maxn = 400 + 4, maxk = 10;

int T, n, k, P[1<<(maxk+1)], P1[1<<(maxk+1)];
LL F[maxn][1<<maxk];

void dfsP(int w, int b) { //搜索所有的长度≤k+1 回文
   assert(w <= k+1);
   if(w == k) { P[b] = 1; return; }
   if(w == k+1) { P1[b] = 1; return; }
   dfsP(w+2, b<<1);                        //两边都加 0
   dfsP(w+2, ((1<<(w+1))+1)|(b<<1));    //两边都加 1
}
```

```
void init() {
    memset(F, -1, sizeof(F));
    memset(P, 0, sizeof(P));
    memset(P1, 0, sizeof(P1));
    dfsP(1, 0);             //1 0 奇数长度回文，中间为0
    dfsP(1, 1);             //1 1 奇数长度回文，中间为1
    dfsP(2, 0);             //2 00 偶数长度回文
    dfsP(2, 3);             //2 11 偶数长度回文
}

//前i位已经决策完成，并且最右边k位为b
int dp(int i, int b) {
    assert(i >= k && i <= n); assert(!P[b]);
    LL& d = F[i][b];
    if(d != -1) return d;
    if(i == n) return d = 1;
    d = 0;
    int nb = b<<1;          //第i+1位为0
    if(!P1[nb] && !P[nb &= ((1<<k)-1)]) d = (d+dp(i+1, nb))%MOD;
    nb = ((b<<1)+1);        //第i+1位为1
    if(!P1[nb] && !P[nb &= ((1<<k)-1)]) d = (d+dp(i+1, nb))%MOD;
    return d;
}

LL pow_mod(LL x, int p) {
    if(p == 0) return 1;
    LL ans = pow_mod(x, p/2);
    ans = (ans * ans) % MOD;
    if(p&1) ans *= x;
    return ans % MOD;
}

int main() {
    cin>>T;
    while(T--) {
        cin>>n>>k;
        if(k > n) { cout<<pow_mod(2, n)<<endl; continue; }
        init();
        LL ans = 0;
        _for(i, 0, (1<<k)) if(!P[i]) ans = (ans + dp(k, i))%MOD;
        cout<<ans<<endl;
```



```
    }
    return 0;
}
```

习题 9-12 保卫 Zonk（Protecting Zonk, ACM/ICPC Dhaka 2006, UVa12093）

给出一个 n（n≤10000）个结点的无根树。有两种装置 A 和 B，每种都有无限多个。

（1）在某个结点 X 使用 A 装置需要 C1（C1≤1000）的花费，并且此时与结点 X 相连的边都被覆盖。

（2）在某个结点 X 使用 B 装置需要 C2（C2≤1000）的花费，并且此时与结点 X 相连的边以及与结点 X 相连的点、相连的边都被覆盖。

求覆盖所有边的最小花费。

【分析】

不难想到是树形 DP，考虑每个结点的覆盖状态。不妨把 n=1 作为树根。令 D(u,s)表示以结点 u 为根的子树，当前覆盖状态为 s，所需的最小花费。对于结点 u，统称其任意孩子为 v，任意孙子为 vx，u 的父亲为 p，p 的父亲为 px，u 的任意兄弟结点统称为 ux。s 分以下 4 种情况（参考图 2.54，虚线表示未覆盖，实线已经覆盖 s）：

（1）s = 0：边 u−v，v−vx，u−p，p−px 全部覆盖。

（2）s = 1：边 u−v，v−vx，u−p 全部覆盖。

（3）s = 2：边 u−v，v−vx 全部覆盖。

（4）s = 3：u−v，还有部分未覆盖。

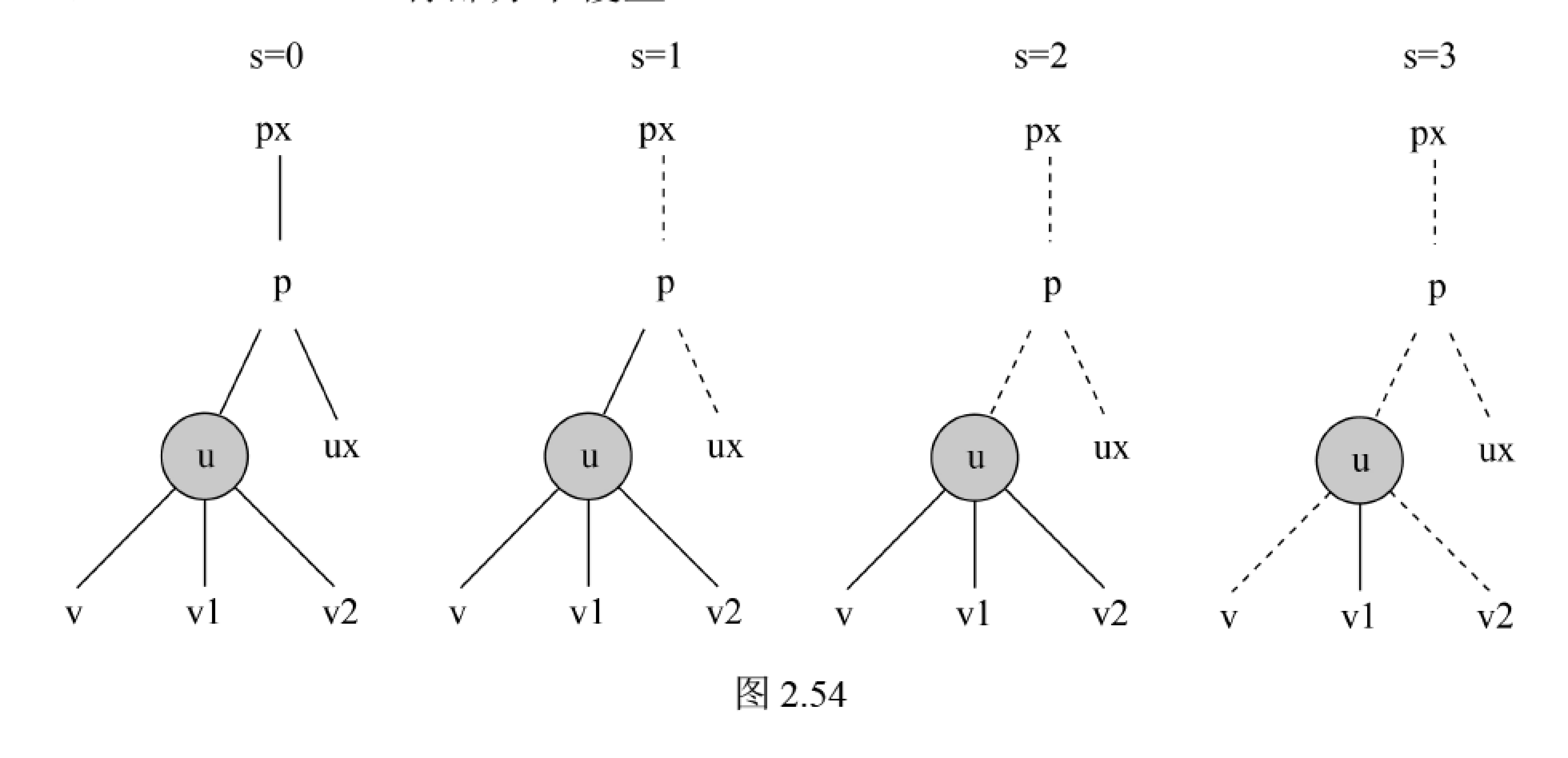


图 2.54

则所求答案为 min(D(1,0), D(1,1),D(1,2))。从叶子到根结点从下往上每次一层进行决策，则这个过程可以写成 dfs，状态转移的前提是 u 的孙子结点对应的子树已经全部覆盖。则状态转移过程如下：

（1）s = 0：这种状态一定要求 u 上放置一个 B，v 的状态转移方程式：D(u,0)= $\sum$(min(D(v,0), D(v,1), D(v,2), D(v,3)))。

（2）s = 1：则要求 v 的覆盖状态≠3。要转移到 u 的这种状态有两种可能：u 上放一个 A，或者且至少有一个 v 的覆盖状态=0。记 w = $\sum$(min(D(v,0), D(v,1), D(v,2)))，则 D(u,1) =

min((C1 + w), w – min(D(v,0) – min(D(v,0), D(v,1), D(v,2)))。

（3）s = 2：要转移到这个状态，就要求所有 v 的覆盖状态为 1。D(u,2) = ∑D(v,1)。

（4）s = 3：则 v 的状态可以是 0、1、2 其中之一。D(u,3) = w。

dfs 中的循环次数刚好是所有的结点次数，所以算法的时间复杂度为 $O(n)$。完整程序（C++11）如下：

```
using namespace std;

const int MAXN = 10004, INF=INT_MAX;
int N, C1, C2, DP[MAXN][4];
vector<int> G[MAXN];
int min(int a, int b, int c){ return min(min(a,b), c);}
int min(int a, int b, int c, int d){ return min(min(a,b), min(c, d));}
void dfs(int u, int fa){
    int *D = DP[u], U1E = 0, minV0 = INF;
    memset(D, 0, sizeof(DP[u]));
    for(auto v : G[u]){
        if(v == fa) continue;
        dfs(v, u);
        int *DV = DP[v], w = min(DV[0], DV[1], DV[2]);
        D[0] += min(DV[0], DV[1], DV[2], DV[3]);
        D[1] += w;                          //u 上面放 A
        D[2] += DV[1];
        D[3] += w;
        U1E += w;                           //u 上面不放 A 的费用
        minV0 = min(minV0, DV[0]-w);        //DV=0 对于费用的增加
    }
    D[0] += C2, D[1] = min(D[1]+C1, U1E + minV0);
}

int main(){
    int u, v;
    while(scanf("%d%d%d", &N, &C1, &C2) == 3 && N){
        _rep(i, 0, N) G[i].clear();
        _for(i, 1, N)
            scanf("%d%d", &u, &v), G[u].push_back(v), G[v].push_back(u);
        dfs(1, -1);
        printf("%d\n", min(DP[1][0], DP[1][1], DP[1][2]));
    }
    return 0;
}
```



习题 9-14　圆和多边形（Telescope, ACM/ICPC Tsukuba 2000, UVa1543）

给出一个圆和圆周上的 n（$3\leqslant n\leqslant 40$）个不同点，请选择其中的 m（$3\leqslant m\leqslant n$）个点，按照在圆周上的顺序连成一个 m 边形，使得它的面积最大。例如在如图 2.55 所示的例子中，右上方的多边形最大。

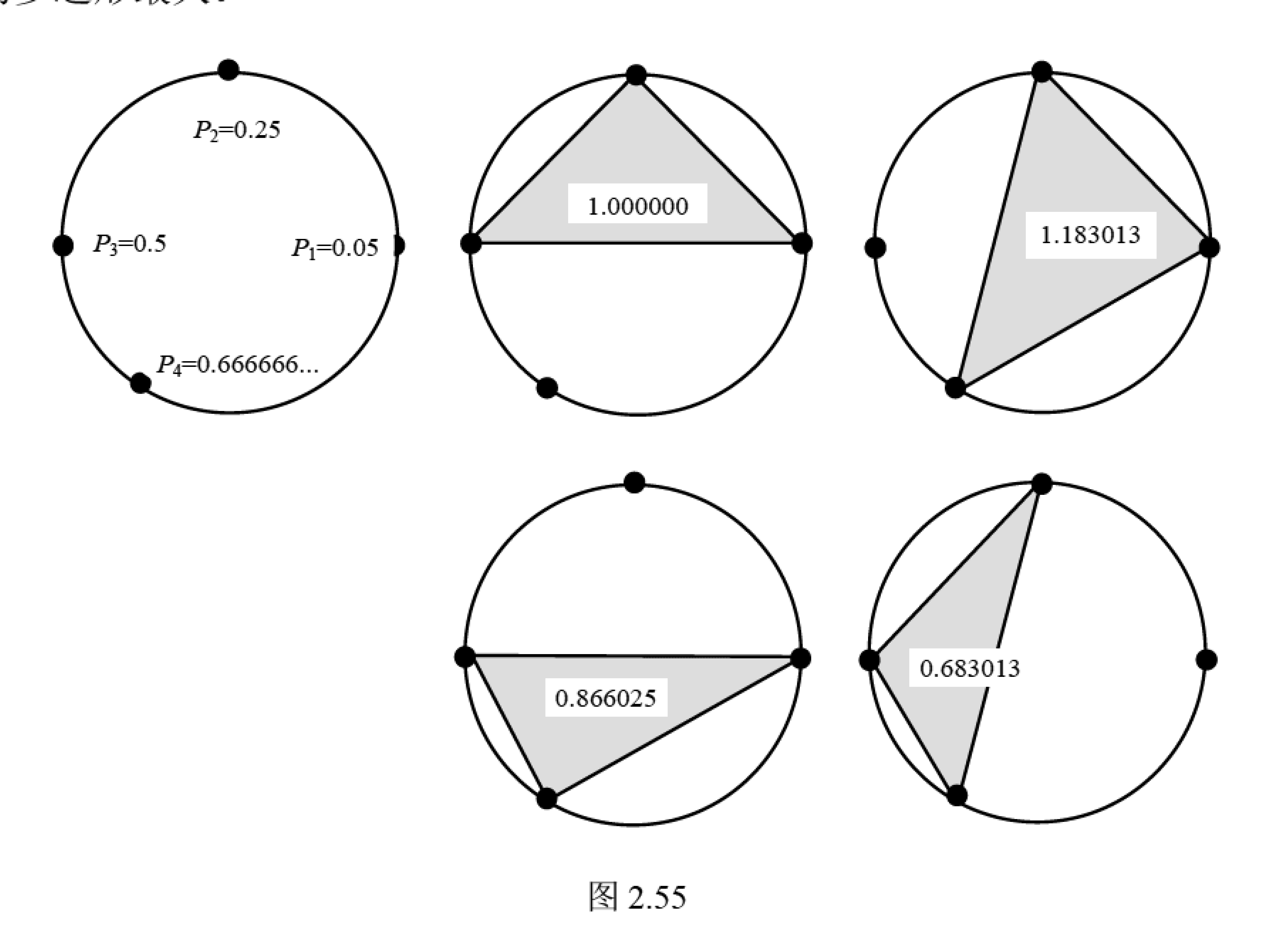


图 2.55

【分析】

假设总共有 n 个点。记 D(i,j,k)为在第 i～j 个点中选择 k 个点（其中必须包含 i 和 j，$0\leqslant i<i+1<j<n$），所能组成的最大的多边形面积。对选择的 k 个点中 j 之前的最后一个点 x（$i<x<j$）进行决策，则有 D(i,j,k) = max(D(i,x,k−1) + area(i,x,j))，如图 2.56 所示。其中，area(i,x,j)为 i、x、j 这 3 个点组成的三角形面积。所求结果为 max(D(0,n,m))。算法的时间复杂度为 $O(n^3)$。

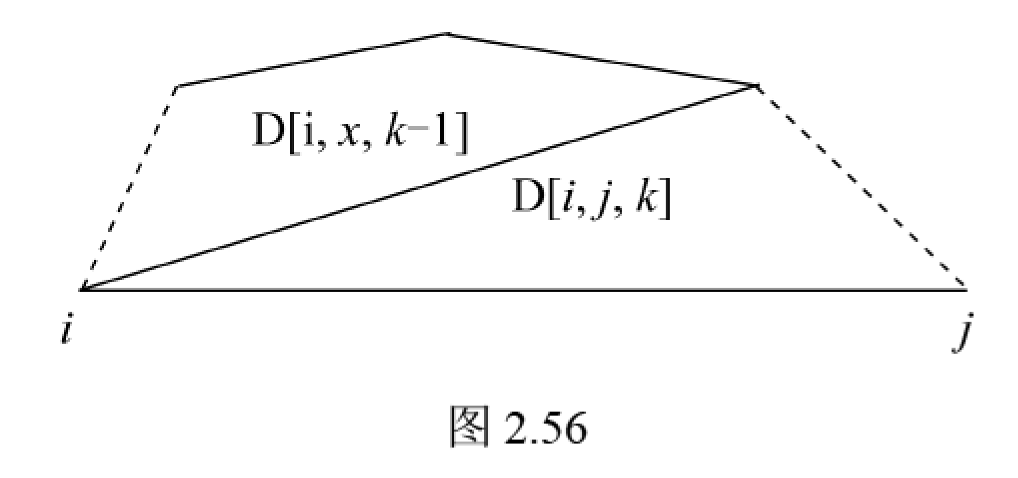


图 2.56

需要注意的是，需要预先把任意 3 个点组成的三角形面积计算出来，以便在递推 D 时使用。完整程序如下：

```
using namespace std;

const int MAXN = 50;
const double PI = acos(-1);
```

```
double dist(double p1, double p2) { //p1,p2 两个角度对应的点的直线距离
   double a = fabs(p2 - p1);        //assert(a < 1);
   if (a > 0.5) a = 1 - a;
   return 2 * sin(a * PI);
}
//3 条边长度为 a、b、c 的三角形面积
double calArea(double a, double b, double c) {
   double x = (a + b + c) / 2;
   return sqrt(x * (x - a) * (x - b) * (x - c));
}

int n, m;
double P[MAXN], d[MAXN][MAXN], area[MAXN][MAXN][MAXN], DP[MAXN] [MAXN] [MAXN];
//dp[i][j][k]表示从第 i 个点到第 j 个点选 k 个点的最大面积（i，j 必须选）
double dp() {
   memset(DP, 0, sizeof(DP));
   _rep(k, 3, m)
      _for(i, 0, n) _for(x, i+1, n) _for(j, x+1, n) // x in [i, j]
         DP[i][j][k] = max(DP[i][j][k], DP[i][x][k-1] + area[i][x][j]);

   double ans = 0;
   _for(i, 0, n) _for(j, i+1, n) ans = max(ans, DP[i][j][m]);
   return ans;
}

int main() {
   while (scanf("%d%d", &n, &m) == 2 && n) {
      _for(i, 0, n) scanf("%lf", &(P[i]));
      //所有点与点之间的距离
      _for(i, 0, n) _for(j, i+1, n) d[i][j] = d[j][i] = dist(P[i], P[j]);
      //计算所有三角形面积
      _for(i, 0, n) _for(j, i + 1, n) _for(k, j + 1, n) {
         area[i][j][k] = area[i][k][j] =
         area[j][i][k] = area[j][k][i] =
         area[k][i][j] = area[k][j][i] =
         calArea(d[i][j], d[j][k], d[k][i]);
      }
      printf("%.6lf\n", dp());
   }
   return 0;
}
```

习题 9-15　学习向量（Learning Vector, ACM/ICPC Dhaka 2012, UVa12589）

输入 n 个向量（x,y）（0≤x,y≤50），要求选出 k 个，从(0,0)开始画，使得画出来的折线与 x 轴围成的图形面积最大。例如 4 个向量是(3,5)、(0,2)、(2,2)、(3,0)，可以依次画(2,2)、



(3,0)、(3,5)，围成的面积是 21.5，如图 2.57 所示。输出最大面积的两倍。$1 \leqslant k \leqslant n \leqslant 50$。

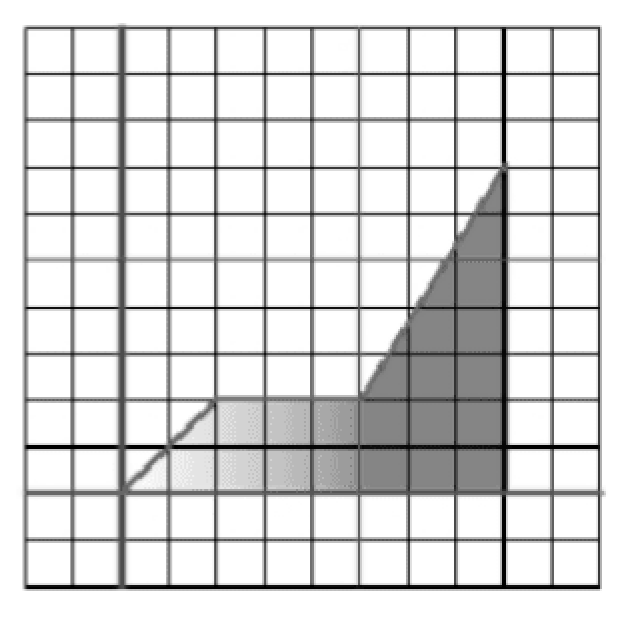


图 2.57

【分析】

如果画出来的折线形成凹多边形，调整成凸多边形一定面积更大（如图 2.58 所示），所以任何最优选择中的 k 个向量一定是按照斜率从大到小依次选择的。对所有向量按照斜率从大到小进行排序。考虑 x 可能为 0，向量的斜率可以使用函数 atan2(y, x)计算。

记 F(i,c,h)为还要在 i 个向量中，还要选择 c 个，画出折线的最高 y 坐标为 h。后续还能再增加的最大面积（如图 2.59 的浅色部分所示）。

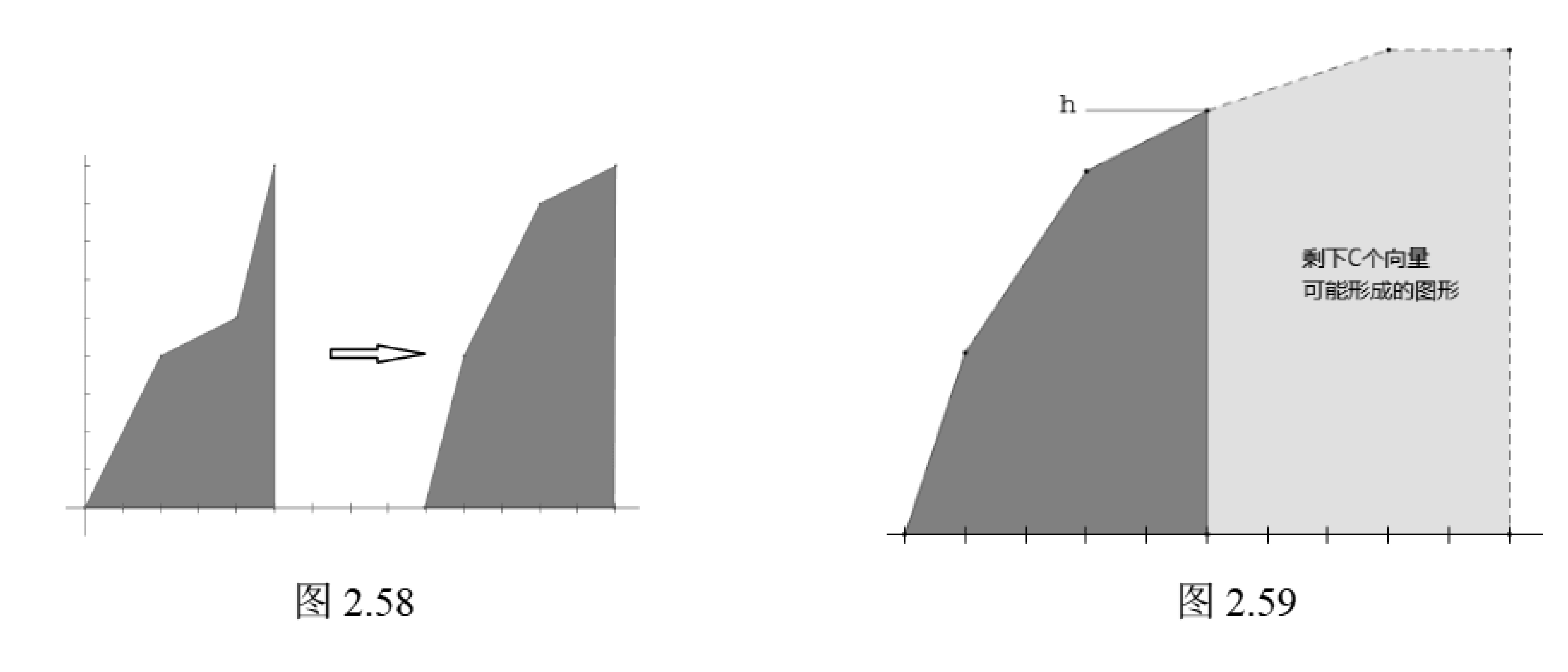


图 2.58　　图 2.59

状态转移方程：F(i,c,y) = max(F(i+1, c, y), F(i+1, c+1, y+v.y)+(2*y + v.y)*v.x)，其中 v 为第 i 个向量，有是否使用向量 v 的两种决策。边界条件为 $i = n$, $c = k$ 时，F = 0。所求结果为 F(0,0,0)。算法的时间复杂度为 $O(50*n^3)$。

习题 9-18　棒球投手（Pitcher Rotation, ACM/ICPC Kaosiung 2006, UVa1379）

你经营着一支棒球队。在接下来的 g+10 天中会有 g（$3 \leqslant g \leqslant 200$）场比赛，其中每天最多一场比赛。你已经分析出你的 n（$5 \leqslant n \leqslant 100$）个投手中每个人对阵所有 m（$3 \leqslant m \leqslant 100$）个对手的胜率（一个 n*m 矩阵），要求给出作战计划（即每天使用哪个投手），使得总获胜场数的期望值最大。需要注意的是，一个投手在上场一次后至少要休息 4 天。

提示：

如果直接记录前 4 天中每天上场的投手编号（1～n），时间和空间都无法承受。

【分析】

每个投手上场一次至少休息 4 天，对于每场比赛来说，胜率最高的 5 个投手不可能前面 4 天都参加比赛（每天最多一场，每场只需 1 个投手），所以至少有 1 个休息够了，可以只考虑胜率最高的 5 个人。

令 DP(i,p0,p1,p2,p3) 表示第 i 天，选择对阵当天对手胜率排名第 p0 的对手上场，且第 i−x 天选择对应排名第 px 的上场，前 i 天所能得到的最大得分。其中 px = 0 则表示当天无人上场。

状态转移时，首先要保证第 i 天选择的对手不能和前面 4 天重复，然后对于每 i 天来说，有两种情况：

（1）没有比赛：则 D(i,0,p1,p2,p3) = max(D(i−1,p1,p2,p3,p4)，其中 0≤p4≤5。

（2）有比赛：则 D(i,p0,p1,p2,p3) = max(D(i−1,p1,p2,p3,p4) + p)，对 p0 进行决策，p 是对当天对手胜率排名 p0 的选手的胜率。

时间复杂度为 $O(g*5^5)$，因为五维数组占用空间较大，需使用滚动数组，这样空间复杂度就是常数 $O(2*6^4)$。完整程序如下：

```
using namespace std;
const int MAXN = 104, MAXG = 200 + 10 + 4;
struct WinP {
   int p, pit;                                     //胜率，以及选手编号
   WinP() : p(0), pit(0) {}
   bool operator<(const WinP& s) const { return p > s.p; }
};

WinP winps[MAXN][MAXN];
int N, M, numG, G[MAXG], DP[2][6][6][6][6];        //滚动数组
//面对对手 op 胜率第 i 高的选手
inline int getPi(int op, int i) { return winps[op][i].pit; }

double solve() {
   memset(DP, 0, sizeof(DP));
   int ans = 0, cur = 0;
   _rep(i, 1, numG) {                              //第 i 天
      int prev = cur, now = 1 - cur;
      cur = 1 - cur;
      memset(DP[now], 0, sizeof(DP[now]));

      int op = G[i];                               //第 i 天面对的对手
      if (op == 0) {                               //当天没有比赛
         _rep(p1, 0, 5) _rep(p2, 0, 5) _rep(p3, 0, 5) _rep(p4, 0, 5) {
            int& d = DP[now][0][p1][p2][p3];      //当天不用派人上场
            d = max(d, DP[prev][p1][p2][p3][p4]); //用前一天的数据更新今天
```

```
                ans = max(d, ans);
            }
            continue;
        }

        _rep(i0, 1, 5) { //今天选排第 i0 的上场
            int opi = getPi(op, i0);             //选上的人是谁
            _rep(p1, 0, 5) {
                if (i > 1 && getPi(G[i-1], p1) == opi) continue;
                _rep(p2, 0, 5) {
                    if (i > 2 && getPi(G[i-2], p2) == opi) continue;
                    _rep(p3, 0, 5) {
                        if (i > 3 && getPi(G[i-3], p3) == opi) continue;
                        _rep(p4, 0, 5) {
                            if (i > 4 && getPi(G[i-4], p4) == opi) continue;
                            int& d = DP[now][i0][p1][p2][p3];
                            d  =  max(d,  DP[prev][p1][p2][p3][p4]  +  winps[op]
[i0].p);
                                ans = max(d, ans);
                        }
                    }
                }
            }
        }
    }

    double d = ans * .01;
    return d;
}

int main() {
    int T; scanf("%d",&T);
    while (T--) {
        scanf("%d%d%d", &N, &M, &numG); numG += 10;
        _rep(i, 1, M){
            _rep(j, 1, N){
                scanf("%d", &(winps[i][j].p));
                winps[i][j].pit = j;
            }
            sort(winps[i] + 1, winps[i] + N + 1);
        }

        G[0] = 0;
        _rep(i, 1, numG) scanf("%d", &(G[i]));
```

```
        double ans = solve();
        printf("%.2lf\n", ans);
    }
    return 0;
}
```

2.8 数学概念与方法

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 10 章。

习题 10-1 砌砖（Add Bricks in the Wall, UVa11040）

对 45 块石头按照如图 2.60 所示的方式排列，每块石头上有一个整数。

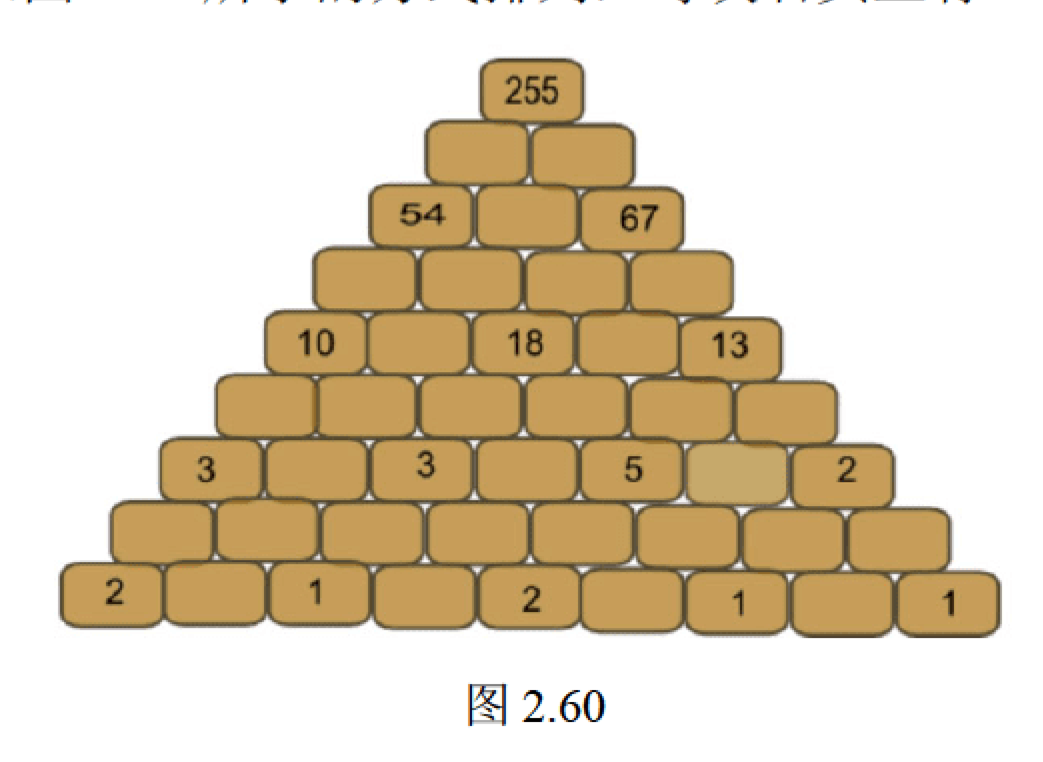


图 2.60

除了最后一行外，每个石头上的整数等于支撑它的两个石头上的整数之和。目前只有奇数行的左数奇数个位置上的数已知，你的任务是求出其余所有整数。输入保证有唯一解。

【分析】

把所有石头看作一个二维数组 B[9][9]。从上到下依次是 0～8 行。则对于 i=0,2,4,6,8 这样的偶数行，已知数字就是 B[0][0]、B[2][0]、B[2][4]这样的偶数列。我们观察第 8 行的左数第 1 个空格，假设里面是 x，则其上方的两个石头分别为 2+x、x+1，继续往上就有 $2+x+x+1=3$，这样就可以解出 x。

推广开来，对于奇数列来说，$B_{i,j}$ 满足：

$$B_{i,j}+B_{i,j-1}=B_{i-1,j-1}$$
$$B_{i,j}+B_{i,j+1}=B_{i-1,j}$$
$$B_{i-1,j}+B_{i-1,j-1}=B_{i-2,j-1}$$

所以有：

$$B_{i,j}=\frac{(B_{i-2,j-1}-B_{i-1,j}-B_{i-1,j-1})}{2}$$

从下到上把所有的偶数行奇数列的数字计算出来之后，奇数行的数字按照题目所述规则也自然可以计算出来。

习题 10-2 勤劳的蜜蜂（Bee Breeding, ACM/ICPC World Finals 1999, UVa808）

如图 2.61 所示，输入两个格子的编号 *a* 和 *b*（*a*,*b*≤10000），求最短距离。例如，19 和 30 的距离为 5（一条最短路是 19-7-6-5-15-30）。

【分析】

在一个平面上任选两个向量，就可以建立一个坐标体系（本题中已经不是笛卡尔坐标系）表示平面上所有的点。如图 2.62 所示，使用块 1→7 的方向作为正 X 轴，1→6 的方向作为正 Y 轴。XY 轴把平面分成 4 个象限，同时用 6 个方向向量表示 6 个方向：{−1,0}, { −1,1}, {0,1}, {1,0}, {1,−1}, {0,−1}。

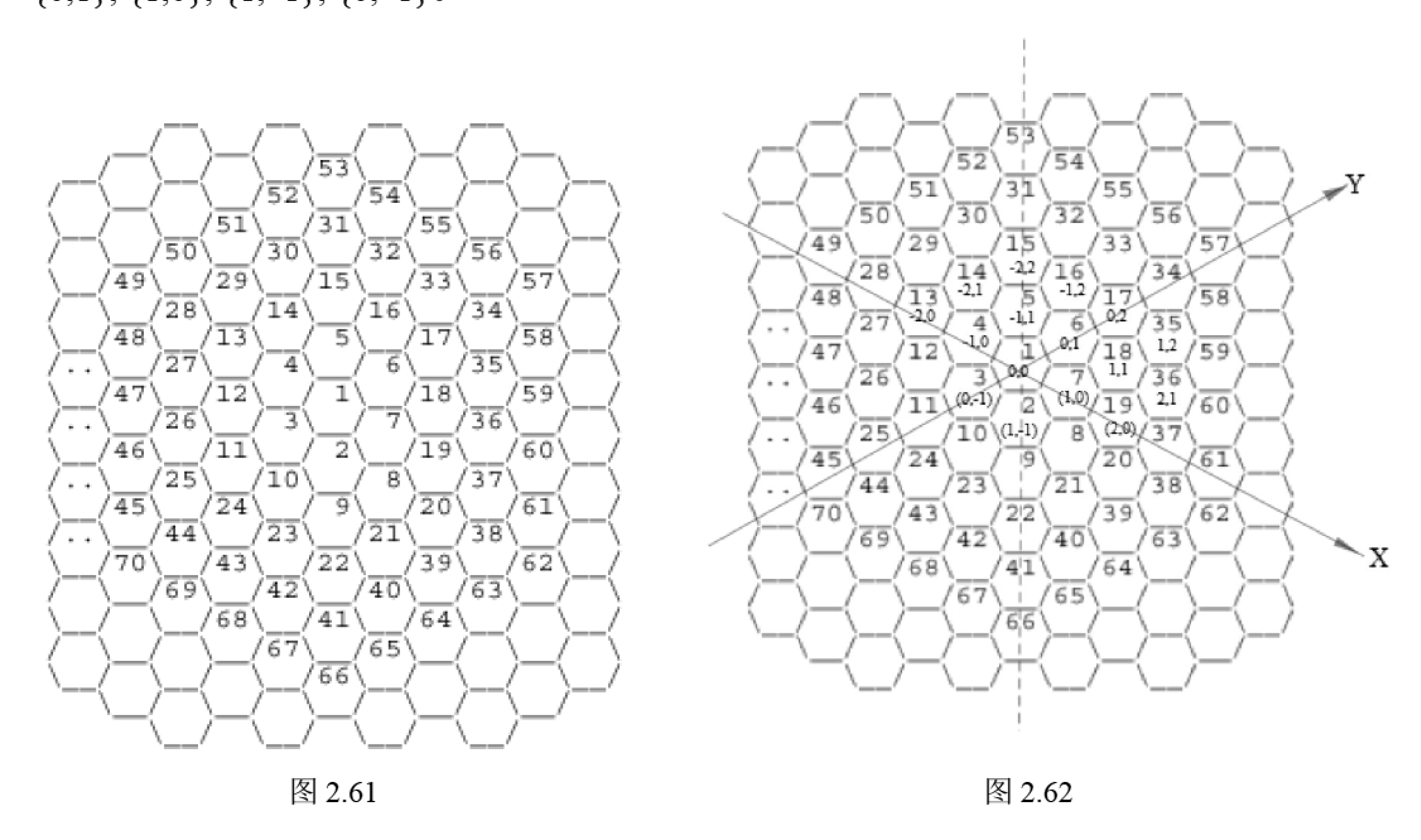


图 2.61　　　　图 2.62

首先确定块 1 和 2 的坐标(0,0)和(1,−1)，观察后可以得出，点的坐标遵循如下规律。

第 1 圈：

2→3, 3→4, 4→5, 5→6, 分别使用向量{−1,0}, { −1,1}, {0,1}, {1,0}, 每次生成 1 个点。

6→8，使用向量{1,−1}，生成 2 个点。

8→9，使用向量{0,−1}，生成 1 个点。

第 2 圈：

9→11, 11→13, 13→15, 15→17, 分别使用向量{−1,0}, { −1,1}, {0,1}, {1,0}, 每次生成 2 个点。

17→20，使用向量{1,−1}，生成 3 个点。

20→22，使用向量{0,−1}，生成 2 个点。

……

依次类推，就可以生成所有点的坐标。对于任意的两个点(x_1, y_1), (x_2, y_2)，记 $x = x_1 - x_2$, $y = y_1 - y_2$，则这个两点之间的距离就是原点到(x,y)的距离。

观察之后不难发现：第 1、3 象限的点(x, y)到原点的距离为 $|x|+|y|$；第 2、4 象限的点(x, y)到原点的距离为 $\max\{|x|, |y|\}$，问题得解。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 10000;
Point pos[MAXN + 330];
vector<Vector> dirs = {{-1,0}, {-1,1}, {0,1}, {1,0}, {1,-1}, {0,-1}};
//6 个方向
int main() {
   int pi = 2;
   pos[pi] = Point(1, -1);
   auto calPos = [&pi](int dir, int l) { //向 dir 方向递推 l 个格子的坐标
      pi++;
      while (l--) {
         pos[pi] = pos[pi - 1] + dirs[dir];
         pi++;
      }
      pi--;
   };

   auto dist = [](const Vector & v) { //计算向量的长度
      if ((v.x < 0 && v.y > 0) || (v.x > 0 && v.y < 0))
         return max(abs(v.x), abs(v.y));
      return abs(v.x + v.y);
   };

   //按照每一圈递推坐标
   _rep(l, 1, 58) { //第 l 个圈
      _for(dir, 0, 4) calPos(dir, l);
      calPos(4, l + 1);
      calPos(5, l);
   }

   int n, m;
   while (scanf("%d%d", &n, &m) == 2 && n)
      printf("The distance between cells %d and %d is %d.\n",
         n, m, dist(pos[n] - pos[m]));

   return 0;
}
```

本题代码参考了 http://www.cnblogs.com/AOQNRMGYXLMV/p/4202527.html。

习题 10-4　素数间隔（Prime Gap, ACM/ICPC Japan 2007, UVa1644）

输入一个整数 n，求它后一个素数和前一个素数的差值。输入是素数时输出 0。n 不超过 1299709（第 100000 个素数）。例如 n=27 时输出 29−23=6。

【分析】

首先筛选出符合条件的所有素数，存在一个 vector<int> primes 中。对于 n 来说，令 pl = lower_bound(primes.begin(), primes.end(), n)。pl 就指向第一个大于等于 n 的素数。如果*pl = N，则说明 N 是素数；否则 pl−1 指向最后一个小于 n 的素数。

习题 10-5　不同素数之和（Sum of Different Primes, ACM/ICPC Yokohama 2006, UVa1213）

选择 K 个质数，使它们的和等于 N。给出 N 和 K（$N \leqslant 1120$，$K \leqslant 14$），问有多少种满足条件的方案？例如 n=24，k=2 时有 3 种方案：5+19=7+17=11+13=24。注意，1 不是素数，因此 n=k=1 时答案为 0。

【分析】

首先需要筛选出所有可能符合条件的素数。之后这个问题就转换成一个背包问题：

（1）$d(i,n,k)$表示从第 i 个及其以后的素数中选择 k 个素数其和为 n 的方案个数。

（2）递推公式就是：$d(i,n,k) = d(i+1,n,k) + d(i+1, n-p_i, k-1)$，分别对应是否使用第 i 个素数 p_i 的策略。

边界条件如下：

（1）$n < 2$ 或者 $p_i > n$ 时，则 $d = 0$。

（2）$k = 1$ 时，如果 n 是素数，则 d=1，否则 d=0。

习题 10-6　连续素数之和（Sum of Consecutive Prime Numbers, ACM/ICPC Japan 2005, UVa1210）

输入整数 n（$2 \leqslant n \leqslant 10000$），有多少种方案可以把 n 写成若干个连续素数之和？例如输入 41，有 3 种方案：2+3+5+7+11+13、11+13+17 和 41。

【分析】

首先筛选出所有符合条件素数组成的序列 P，然后求出 P 的前缀和序列 PS，则所求结果就是符合条件“PS[i] + n 依然在 PS 中”的 i 的个数。

习题 10-7　几乎是素数（Almost Prime Numbers, UVa10539）

输入两个正整数 L、U（$L \leqslant U \leqslant 10^{12}$），统计区间[$L$,$U$]的整数中有多少个数满足：它本身不是素数，但只有一个素因子。例如 4 和 27 都满足条件。

【分析】

这种数的唯一分解中一定只包含一个素数：p^k，其中 $k \geqslant 2$，记 $n = \sqrt{10^{12}}$，则显然有 $p \leqslant n$。我们对素数筛法进行改造：每发现一个素数 p 就把所有的 p^k（$k \geqslant 2$ 且 $p^k \leqslant 10^{12}$）记录下来。最后按照从小到大的顺序把这些数字记录下来存放到一个 vector 中，记为 aps。

对于区间[L,U]，第一个大于等于 L 的数就是 lower_bound(aps.begin(), aps.end())，第一

个大于 U 的数就是 upper_bound(aps.begin(), aps.end(), U)，则这两个位置形成一个左闭右开区间，区间中数字的个数即是所求的结果。

和《算法竞赛入门经典（第 2 版）》中的第 10.1.2 节类似，虽然内循环多了一个步骤，但这个步骤的循环次数的最大值为 $\log_p(10^{12}) \approx 39$，整体的算法复杂度为 $O(n\log n)$。完整程序（C++11）如下：

```
using namespace std;
typedef long long LL;
const LL MAXN = 1000000 + 10, MAXP = 1000000000000;
vector<LL> aps; //almost primes

void sieve() {
   vector<bool> vis(MAXN, false);
   aps.reserve(MAXN);
   for(LL i = 2; i < MAXN; i++) if(!vis[i]){
      for(LL j = i*i; j < MAXN; j += i) vis[j] = true;
      for(LL p = i*i; p <= MAXP; p *= i) aps.push_back(p);
   }

   sort(aps.begin(), aps.end());
}

int main(){
   sieve();
   int N; scanf("%d", &N); LL L, H;
   while(N--) {
      scanf("%lld%lld", &L, &H);
      auto pL = lower_bound(aps.begin(), aps.end(), L),
         pH = upper_bound(aps.begin(), aps.end(), H);
      printf("%ld\n", pH - pL);
   }
   return 0;
}
```

习题 10-8　完全 P 次方数（Perfect Pth Powers, UVa10622）

对于整数 x，如果存在整数 b 使得 $x=b^p$，我们说 x 是一个完全 p 次方数。输入整数 n，求出最大的整数 p，使得 n 是完全 p 次方数。n 的绝对值不小于 2，且 n 在 32 位带符号整数范围内。例如 n=17, p=1；n=1073741824, p=30；n=25, p=2。

【分析】

对于 x 是正数的情况，首先求出 x 的唯一分解 $p1^{k1}p2^{k2}$、…、pn^{kn}。如果 x 是一个全 p 次方数，则所有的 ki 必然都是 p 的倍数，然后求所有 ki 的最大公约数就是 p。



需要注意的是，如果 x 是负数，则 p 不能是偶数，所以求出所有 ki 的最大公约数之后，还要把其中的 2 除尽才得到 p。

习题 10-9 约数(Divisors, UVa294)

输入两个整数 L 和 U（$1 \leqslant L \leqslant U \leqslant 10^9$，$U-L \leqslant 10000$），统计区间$[L,U]$的整数中哪一个的正约数最多。如果有多个，输出最小值。

【分析】

对于整数 K 来说，假设其唯一分解为$\prod p_i^{ki}$，则约数个数为$\prod k_i$。可以用筛法先求出所有 1～10^5之间的素数，然后对 L, U 之间的整数进行唯一分解并且遍历，即可求出结果。

习题 10-10 统计有根树（Count, Chengdu 2012, UVa1645）

输入 n（$n \leqslant 1000$），统计有多少个 n 结点的有根树，使得每个深度中所有结点的儿子数相同。例如 n=4 有 3 棵，如图 2.63 所示；n=7 时有 10 棵。输出数目除以 10^9+7 的余数。

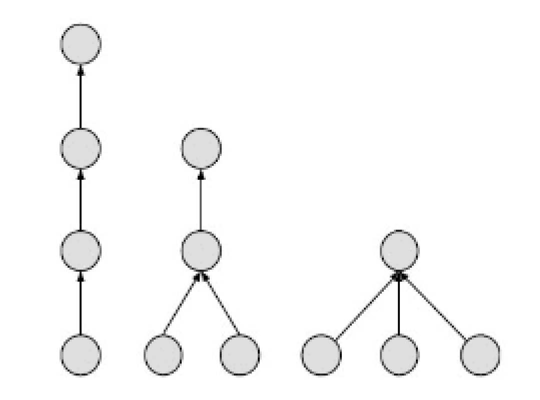


图 2.63

【分析】

令 A[n]为所求的值，首先对于 n=1,2 来说 A[n] = 1。根据题意，一个结点的所有子树结构都必须完全一致，则对 n 个结点数的子树的结点个数 j 进行遍历就可以得出：A[n] = $\sum$A[j]，其中 $n-1$ 能被 j 整除。注意要使用 long long，防止数据溢出。

习题 10-11 圈图的匹配（Edge Case, ACM/ICPC NWERC 2012, UVa1646）

n（$3 \leqslant n \leqslant 10000$）个结点组成一个圈，求匹配（即没有公共点的边集）的个数。例如 n=4 时有 7 个（如图 2.64 所示），n=100 时有 792070839848372253127 个。

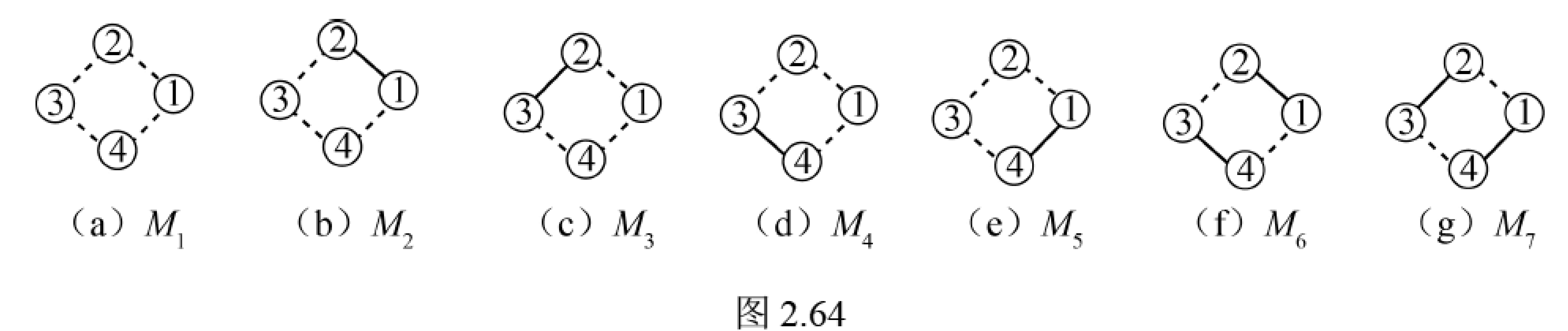


图 2.64

【分析】

首先看另一个问题，令 F[n]为 n 个点组成线段链（不连成圈）中匹配的个数，则 F[1] = 1，F[2] = 2，F[3] = 3。考虑第 $n-1$ 个线段是否在匹配中：如果在，则线段 $n-2$ 不在，故有 F[$n-2$]个匹配；否则有 F[$n-1$]个匹配。由此可得 F[n] = F[$n-1$] + F[$n-2$]，其实就是斐波那契数列。

回到本题，记所求匹配数为 C[n]，根据点 1 和点 2 之间的线段是否在匹配中两种情况，可得 C[n] = F[n] + F[$n-2$]。递推计算即可。注意，本题中的结果连 long long（64 位整数）也放不下，需使用大整数类。

习题 10-12 汉堡（Burger, UVa557）

有 n 个牛肉堡和 n 个鸡肉堡给 $2n$ 个孩子吃。每个孩子在吃之前都要抛硬币，正面吃牛肉煲，反面吃鸡肉煲。如果剩下的所有汉堡都一样，则不用抛硬币。求最后两个孩子吃到

相同汉堡的概率。

【分析】

在正面或者背面的次数达到 n 次之后，就会出现符合题意的条件。但是情况太多，分类计数很麻烦，从另外一个角度来考虑，如果扔的前 $2n-2$ 次，正面和反面出现的次数一样多，则最后两个孩子就会吃到不同的汉堡，记这种情况出现的概率为 F_n，则所求结果为 $1-F_n$，且有 $F_n=\frac{C_{2n-2}^{n-1}}{2^{2n-2}}$。但是分式的上下两项直接计算的话，必然会溢出。可从上述公式进一步得到 F_n 的递推公式：$F_1=1, F_n=\frac{(2n-3)}{2n-2}F_{n-1}$，直接递推计算即可。

习题 10-13　H(n)（H(n), UVa11526）

输入 n（在 32 位带符号整数范围内），计算下面 C++函数的返回值：

```
long long H(int n){
    long long res = 0;
    for( int i = 1; i <= n; i=i+1 ){
        res = (res + n/i);
    }
    return res;
}
```

例如 n=5, 10 时，答案分别为 10 和 27。

【分析】

以 $n=10$ 为例，i=1～10 时累加的数字依次是 10, 5, 3, 2, 2, 1, 1, 1, 1, 1。有 5 个 1，2 个 2，1 个 3…。

由此考虑 n/i=1 时，i 的范围，显然是 $n/2<i\leqslant n/1$，就有（$n/1-n/2$）个 1。

n/i=2 时，$n/3<i\leqslant n/2$，就有（$n/2-n/3$）个 2。

n/i=3 时，$n/4<i\leqslant n/3$，就有（$n/3-n/4$）个 3。

…

n/i=k 时，$n/(k+1)<i\leqslant n/k$，有（$n/(k+1)-n/k$）个 k。

需要注意的是，当 $\frac{n}{k}-\frac{n}{k+1}=1$ 开始，就按照正常的 n/i 计算。此时有 $\frac{n}{k}-\frac{n}{k+1}=1\rightarrow n\approx k(k+1)\rightarrow k\approx\sqrt{n}$。算法的时间复杂度就是 $O(\sqrt{n})$。主循环逻辑如下：

```
LL solve(LL n){
    LL ans = 0, i = 0;
    for(i = 1; i <= n; i++){
        LL c = n/i - n/(i+1);
        ans += i*c;
        if(c <= 1) break;
    }
```

```
    for(i = n/(i+1); i >= 1; i--) ans += n/i;
    return ans;
}
```

注意 n<0 时循环不会运行，直接返回 0。

习题 10-15 零和一（Zeros and Ones, ACM/ICPC Dhaka 2004, UVa12063）

给出 n, k（$n\leqslant 64$，$k\leqslant 100$），有多少个 n 位（无前导 0）二进制数的 1 和 0 一样多，且值为 k 的倍数？

【分析】

使用 $D(z,o, m)$表示符合以下条件的数字的个数：

（1）以 1 开头。

（2）1 后面共有 z 个 0，o 个 1。

（3）除 k 的余数为 m。

则边界条件为 $D(0,0,1) = 1$。每个二进制数后面每附加一个数字就有两种转移方式：

（1）$D(z+1,o, (m*2)\%k)$ += $D(z,o,m)$ //后面附加一个 0

（2）$D(z,o+1, (m*2+1)\%k)$ += $D(z,o,m)$ //后面附加一个 1

当 n 为奇数或 k=0 时无解。当 n 是偶数时，所求答案为：$D(n/2, n/2-1,0)$。

习题 10-16 计算机变换（Computer Transformations, ACM/ICPC SEERC 2005, UVa1647）

初始串为一个 1，每一步会将每个 0 改成 10，每个 1 改成 01，因此 1 会依次变成 01, 1001, 01101001, … 输入 n（$n\leqslant 1000$），统计 n 步之后得到的串中，“00”这样的连续两个 0 出现了多少次。

【分析】

我们分析所有可能的变换：

0→10

1→01

00→1010

10→0110

01→1001

11→0101

记 F_n 为第 n 步之后的 00 的个数，由前文所示，00 是由 01 得到的，只需知道 $n-1$ 步后 01 的个数 E_{n-1} 就得到 F_n。再看前文推导，01 由 1 和 00 得到，而第 n 步后 1 的个数是 2^{n-1}，所以 $F_n=E_{n-1}=2^{n-3}+F_{n-2}$，由此直接递推即可。注意 $n\leqslant 1000$，所以必须使用高精度整数运算。

习题 10-17 H-半素数（Semi-prime H-numbers, UVa11105）

所有形如 $4n+1$（n 为非负整数）的数叫 H 数。我们定义 1 是唯一的单位 H 数，H 素数是指本身不是 1，且不能写成两个不是 1 的 H 数的乘积。H-半素数是指能写成两个 H 素数的乘积的 H 数（这两个数可以相同，也可以不同）。例如 25 是 H-半素数，但 125 不是。

输入一个 H 数 h（$h\leqslant 1000001$），输出 1 到 h 之间有多少个 H-半素数。

【分析】

参考素数筛法，用 vis[i]记录 $4i+1$ 是否是 H-素数。对于每个整数 i（$4i+1$ 不是 H-素数），令 $h_i=4*i+1$，对于所有的 $j=(k*h_i)*h_i$（$k\geqslant h_i$），如果 $j\%4=1$，则说明 j 不是 h 素数，记 vis[$(j-1)/4$]=1。否则记录 h_i 为 H-素数。筛法部分外层循环的次数为 n，给定外层的循环变量，内层循环的次数小于 $\frac{4n+1}{4i+1}<\frac{n+1}{i+1}$。所以参考《算法竞赛入门经典（第 2 版）》10.1.2 节的时间复杂度分析可以得出，本题筛法部分的时间复杂度上限依然为 $O(n\log n)$。

筛出所有的 H 素数之后，两两相乘将所有的乘积记录下来，注意可能重复，然后用一个 cnt[N]数组记录 H 半素数的个数，其中 cnt[i]表示 1 到 $4i+1$ 之间所有的 H 半素数的个数，用 $O(n)$的时间就可以将 cnt 处理完成。这样问题的解就是 cnt$\left[\frac{h-1}{4}\right]$。完整程序如下：

```
using namespace std;
typedef long long LL;
const LL MAXN = 250000, MAXP = 4*MAXN+1;
vector<LL> primes;
int cnt[MAXN + 1];                  //cnt[i]表示[1,i]区间内所求素数的种类个数
void sieve() {
   vector<int> vis(MAXN + 1, 0);    //vis[i] → 4i+1 是否是 H-素数
   for(LL i = 1; i <= MAXN; i++) if(!vis[i]){
      LL hi = 4*i + 1;
      for(LL j = hi*hi; j <= MAXP; j += hi)
         if(j%4 == 1) vis[(j-1)/4] = 1;   //j 不是 H-素数
      primes.push_back(hi);
   }

   _for(i, 0, primes.size()) _for(j, i, primes.size()){
      LL hi = primes[i]*primes[j];
      if(hi > MAXP) break;
      vis[(hi-1)/4] = 2;            //hi 是 H-半素数
   }

   cnt[0] = 0;
   _rep(i, 1, MAXN) cnt[i] = cnt[i-1] + ((vis[i] == 2)?1:0);
}

int main(){
   sieve();
   int h;
   while(scanf("%d", &h) == 1 && h) printf("%d %d\n", h, cnt[(h-1)/4]);
   return 0;
}
```

习题 10-18 一个研究课题（A Research Problem, UVa10837）

输入正整数 m（$m \leqslant 10^8$），求最小的正整数 n，使得 $\varphi(n)=m$。输入保证 n 小于 200000000。

【分析】

记 N=200000000。假设 p_1，$p_2 \cdots p_k$ 为 n 的素因子，则 $\varphi(n) = p_1^{m_1}(p_1-1)p_2^{m_2}(p_2-1) \cdots p_k^{m_k}(p_k-1)$，注意其中的 m_i 有可能为 0。且如果 $p_k > \sqrt{n}$，m_k 一定为 1。

首先筛选出所有小于 $\sqrt{N}$ 的素数。然后对于 n，遍历所有不大于 $\sqrt{n}$ 的素数 p，如果 $\varphi(n)$ 能被 $p-1$ 整除，则 p 有可能是 n 的素因子，但出现在 n 的唯一分解中的次数未知。

然后对所有可能是 n 的素因子的 p 的次数进行回溯，每一步尝试 p 的次数为 0,1,2…，在 $\varphi(n)$ 除去相应的 p^m 以及 $(p-1)$。最后当所有可能素数决策完之后，$\varphi(n)$ 还可能没除干净，剩余一个 x，此时就要判断 $x+1$ 是未被使用过的素数，如果是，则说明我们得到了一个合法的 n。完整程序（C++11）如下：

```
using namespace std;
const int MAXP = 14143, INF = 200000000 + 1;   //sqrt(200000000) + 1
typedef long long int64;
vector<int> primes, isPrime(MAXP, 0);          //primes
void sieve() {
   _for(i, 2, MAXP) if(!isPrime[i]) {
      for(int j = i*i; j < MAXP; j += i) isPrime[j] = 1;
      primes.push_back(i);
   }
}
//Φ(n) = phi 时，所有可能是 n 的素因子的数字，存放到 ps 中
void getPrimeFactors(int phi, vector<int>& ps) {
   ps.clear();
   for(auto p : primes){
      if(p > phi) break;
      if(phi%(p-1) == 0) ps.push_back(p);
   }
}
//可能的素因子，决策过的素因子个数，使用的素因子，目前使用的 p 组成的 n，除剩下的 phi
void dfs(const vector<int>& ps, int cur, set<int>& usedPs, int n, int rem,
int& ans) {
      //printf("cur == %d, usedPs=%s n = %d, rem = %d\n", cur, toString
(usedPs).c_str(), n, rem);
      if(cur == ps.size()) {
         if(rem == 1) { ans = min(ans, n); return; } //phi 被除尽
         bool r = true;
         int pr = rem+1;
         for(auto p : primes) { //判断 rem+1 是不是素数
```

```
            if(p*p > pr) break;
            if(pr%p == 0) { r = false; break; } //p不是素数
        }

        //rem+1是没有用过的素数
        if(r && usedPs.count(pr) == 0) ans = min(ans, n*pr);
        return;
    }

    int p = ps[cur];
    //不用p作为n的因子
    dfs(ps, cur+1, usedPs, n, rem, ans);

    if(rem % (p-1)) return;     //不是n的因子，否则尝试用p作为n的因子
    rem /= p-1, n *= p;
    usedPs.insert(p);
    while(true) { //尝试各种次方
        dfs(ps, cur+1, usedPs, n, rem, ans);
        if(rem%p) break;

        assert(rem >= p);
    }
    usedPs.erase(p);
}
int solve(int phi) {
    vector<int> ps;
    set<int> usedPs;
    getPrimeFactors(phi, ps);
    int ans = INF;
    dfs(ps, 0, usedPs, 1, phi, ans);
    return ans;
}
int main() {
    sieve();
    for(int phi, t = 1; scanf("%d", &phi) == 1 && phi; t++)
        printf("Case %d: %d %d\n", t, phi, solve(phi));
    return 0;
}
```

习题 10-19　蹦极（Bungee Jumping, UVa10868）

007 为了摆脱敌人的追击，逃到了一座桥前。桥上正好有一条蹦极绳，于是他打算把它拴到腿上，纵身跳下桥，落地后切断绳子，继续逃。已知绳子的正常长度为 l，Bond 的体重为 w，桥的高度为 s，你的任务是替 007 判断能否用这种方法逃生。



当从桥上跳下后，绳子绷紧前 Bond 将做自由落体运动（重力按 9.81w 计），而绷紧后绳子会有向上的拉力，大小为 $k*\Delta l$，其中 Δl 为绳子当前长度和正常长度之差。当且仅当 Bond 可以到达地面，且落地速度不超过 10 米/秒时，我们才认为他安全着落。

输入每组数据包含 4 个非负整数 k、l、s、w（s<200）。对于每组数据，如果可以安全着地，输出“James Bond survices.”，如果到不了地面，输出“Stuck in the air.”，如果到达地面速度太快，输出“Killed by the impact.”。

【分析】

弹簧弹力和 Δl 的关系是一个三角形（如图 2.65 所示），弹簧伸长的长度从 0 到 Δl，对 Bond 做的功刚好是这个三角形的面积，即 $k*\Delta l^2/2$。

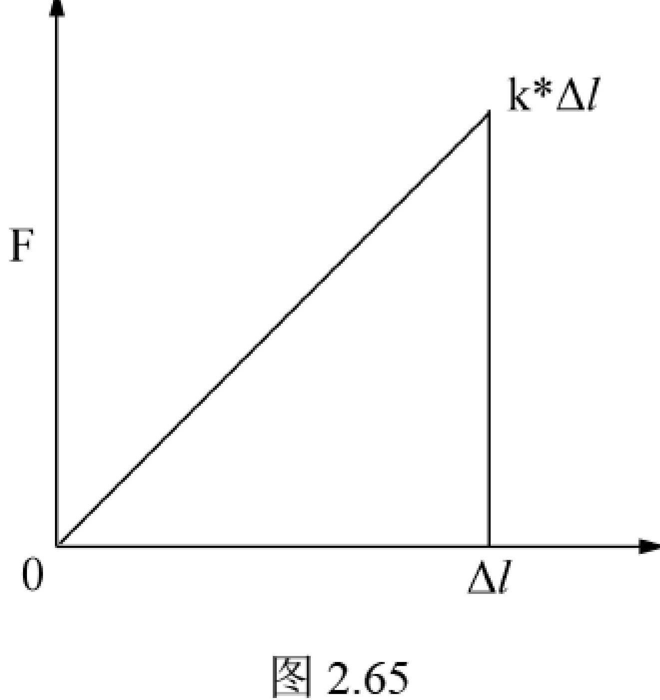


图 2.65

记 v 为触底的速度，根据动能定理，合外力对物体所做的功，等于物体动能的变化，$wgs-\frac{kL^2}{2}=\frac{wv^2}{2}$，因此有 $v^2=2gs-\frac{kL^2}{w}$，其中 $L=s-1$，$s<1$ 时 $L=0$。针对 v^2 的值分情况讨论：

（1）如果 $v^2\leqslant 0$，说明卡在半空。

（2）如果 $v^2\leqslant 100$，说明安全着陆。

（3）否则说明触底速度大于 10，摔死了。

习题 10-20　商业中心（Business Center, NEERC 2009, UVa1648）

商业中心是一幢无限高的大楼。在一楼有 m 座电梯，每座电梯只有两个键：上、下。对于第 i 座电梯，每按一次“上”会往上走 u_i 层楼，每按一次“下”会往下走 d_i 层楼。你的任务是从一楼开始选一个电梯，恰好按 n 次按钮，到达一个尽量低（一楼除外）的楼层。中途不能换乘电梯。$1\leqslant n\leqslant 1000000$，$1\leqslant m\leqslant 2000$，$1\leqslant u_i$，$d_i\leqslant 1000$。

【分析】

不妨设对于每一个电梯，向上按了 a 次，向下按了 b 次，则 $a+b=n$，则这个电梯最终停靠的层数就是 $m=a*u-b*d=a*(u+d)-n*d$。本题就是要对每个电梯求 m 的最小正整数值。这个最小值，就是在 $a=n*d/(u+d)+1$ 时取得，这里的除法就是整数除法。

最终的答案就是所有电梯的 m 的最小值。注意，此题需使用 long long。

习题 10-23　Hendrie 序列（Hendrie Sequence, UVa10479）

Hendrie 序列是一个自描述序列，定义如下：

（1）H(1)=0。

（2）如果把 H 中的每个整数 x 变成 x 个 0 后面跟着 $x+1$，则得到的序列仍然是 H（只是少了第一个元素）。

因此，H 序列的前几项为 0,1,0,2,1,0,0,3,0,2,1,1,0,0,0,4,1,0,0,3,0,...。输入正整数 n（$n<2^{63}$），求 H(n)。

【分析】

可以将 H 序列分块，写出 H 序列的前几块（第 0 块特殊），如表 2.2 所示。

表 2.2

块（m）	序列元素	下标 2^{m-1} ～ 2^m-1	块长度（2^{m-1}）
0	0	0	1
1	1	1	1
2	0 2	2～3	2
3	1 0 0 3	4～7	4
4	0 2 1 1 0 0 0 4	8～15	8
5	1 0 0 3, 0 2, 0 2, 1, 1, 1, 0 0 0 0, 5	16～31	16
	...		

同时可以发现第 m 块由以下序列组成：

1 个 第 m-2 块

2 个 第 m-3 块

...

m-1 个 第 0 块

m

根据以上规律，对于一个输入数字 n，首先让 n=n-1（因为使用以 0 为起始下标统计更方便）。然后根据上述规律，找到 n 所在的那个块 m，并且找到 n 在 m 块中的位置。如果正好是块结尾，直接返回 m 即可。如果不是，找到 n 所属的子块以及在子块中的位置，递归计算即可。完整程序如下：

```
using namespace std;
typedef unsigned long long ull;
const ull u1 = 1;

//block size of m
inline ull bsz(ull m) { return m==0 ? 1 : (u1<<(m-1)); }
ull solve(ull b, ull n) { //寻找第 b 个 block 的第 n 个数字
   if(b == 0 || b == 1) { assert(n == 1); return b; }
   if(n == bsz(b)) return b;
   for(ull i = 1; i < b; i++) {
      ull k = b-i-1, sz = bsz(k); //i 个 block k
      if(n <= i*sz) return n%sz ? solve(k, n%sz) : solve(k, sz);
      n -= i*sz;
   }
   return 0;
}
```



```
ull solve(ull n) {
    if(n == 0) return 0;
    ull b = 1; //属于哪个块的
    while(true) {
        ull sz = bsz(b);
        if(sz < n) n -= sz;
        else return solve(b, n);
        b++;
    }
}

int main(){
    ull n;
    while(cin>>n && n) cout<<solve(n-1)<<endl;
    return 0;
}
```

习题 10-28 数字串（Number String, ACM/ICPC Changchun 2011, UVa1650）

每个排列都可以算出一个特征，即从第二个数开始每个数和前面一个数相比是增加（I）还是减少（D）。例如{3,1,2,7,4,6,5}的特征是 DIIDID。输入一个长度为 n−1（2≤n≤1001）的字符串（包含字符 I、D 和?），统计 1～n 有多少个排列的特征和它匹配（其中?表示 I 和 D 都符合）。输出答案除以 1000000007 的余数。

【分析】

参考 LCS 等字符串相关的 DP 问题，不难想到把排列长度以及最后一位数字考虑进去作为状态考虑：记 dp(i,j)为由数字 1～i 组成的排列中，以 j 结尾的符合指定特征的排列个数。但是确定最后一位之后发现，前面的 1～i−1 位并不是 1～i−1 组成的排列，而是{1,2…j−1, j+1…i}组成的排列，无法进行状态转移。而且 n 又比较大，无法简单地用一个位向量来表示这个排列。

观察{3,1,2,7,4,6,5}，将其中所有大于 4 的数字加 1，转换成由{1～4,6～8}组成的排列{3,1,2,8,4,7,6}，其特征字符串依然是 DIIDID。不难由此推广出以下结论：给定一个长度为 i 的特征字符串，由{1,2,…,i}组成的匹配排列和由{1,2,…j−1,j+1,…,i+1}组成的匹配排列一一对应。这样就可以进行先进行等价转换，再状态转移。

按照 i = 1～n 的顺序对第 i 位的数字进行决策，决策时，把{1,2,…j−1,j+1,…i}的排列转换为{1,2…i−1}的排列，则状态转移可以分 3 种情况来讨论：

（1）s[i] == 'I'，则第 i−1 位必须小于 j，于是有 dp(i,j) = dp(i−1,1)+ dp(i−1,2) +…+ dp(i−1,j−1) = $\sum_{k=1}^{j-1} dp(i-1,k)$ 。

（2）s[i] == 'D'，则 j 之前就是以 j+1～i 其中之一为结尾的{1,2…j−1, j+1…i}的排列，也就是以 j～i−1 为结尾的{1,2…i−1}的排列，dp(i, j) = dp(i−1, j) + dp(i−1, j+1) +…+ dp(i−1,i−1) = $\sum_{k=j}^{i-1} dp(i-1,k)$ 。

（3）s[i] == '?'，则 $\mathrm{dp}(i,j) = \mathrm{dp}(i-1,1) + \mathrm{dp}(i-1,2) + \cdots + \mathrm{dp}(i-1,i-1) = \sum_{k=1}^{i-1} dp(i-1,k)$。

观察以上的状态转移方程，不难想到引入 $\mathrm{sum}(i,j)\sum_{k=1}^{j}\mathrm{dp}(i,k)$，则上述的状态转移就简化成：

（1）s[i] == 'I'，$\mathrm{dp}(i,j) = \mathrm{sum}(i-1,j-1)$。

（2）s[i] == 'D'，$\mathrm{dp}(i,j) = \mathrm{sum}(i-1,i-1)-\mathrm{sum}(i-1,j-1)$。

（3）s[i] == '?'，$\mathrm{dp}(i,j) = \mathrm{sum}(i-1,i-1)$。

初始条件为 sum(1,1)=dp(1,1)=1，剩下的按照 i 从小到大，j 从 1 到 i，依次求 dp(i,j)之后更新 sum(i,j)。这样通过引入 sum(i,j)，时间复杂度就从 $O(n^3)$变成 $O(n^2)$。完整程序如下：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
#define _rep(i,a,b) for( int i=(a); i<=(b); ++i)
using namespace std;
typedef long long LL;
const int MAXN = 1024;
const LL MOD = 1000000007;
string Sig;
LL DP[MAXN][MAXN], SUM[MAXN][MAXN];

int main(){
   while(cin>>Sig) {
      int n = Sig.size() + 1;
      SUM[1][1] = DP[1][1] = 1;
      _rep(i, 2, n) _rep(j, 1, i)  {
         char c = Sig[i-2]; LL& d = DP[i][j];
         if(c == 'I') d = SUM[i-1][j-1];
         else if(c == 'D') d = SUM[i-1][i-1] - SUM[i-1][j-1];
         else d = SUM[i-1][i-1];
         SUM[i][j] = (SUM[i][j-1] + d) % MOD;
      }
      cout<<(SUM[n][n]+MOD)%MOD<<endl;
   }
   return 0;
}
```

习题 10-37　倍数问题（Yet Another Multiple Problem, Chengdu 2012, UVa1653）

输入一个整数 n（1≤n≤10000）和 m 个一位十进制数字，找 n 的最小倍数，其十进制表示中不含这 m 个数字中的任何一个。

提示：

需要建一张图，结点 i 代表除以 n 的余数等于 i。巧妙地利用第 6 章学过的 BFS 树可以简洁地解决这个问题。



【分析】

简单的暴力做法，就是从最左边开始逐位使用所有的可用数字尝试构造，看看能不能整除 n。但是这样算法复杂度是指数级的，必然会超时。仔细观察不难发现，每构造一位，除 n 的余数就会发生变化。把模 n 的每个余数作为一个结点，每构造一位就是沿着这张图走一步，则本题实际上就是求到结点 0 的最短路径，使用 BFS 非常合适。

具体来说，建一张图，结点 i 代表除以 n 的余数等于 i，然后每次附加一位数字就进行状态转移，转移是基于模运算：$(10*a+b)\bmod n = (a \bmod n)*10 + b \bmod n$。那么主要的搜索逻辑就是从可用数字开始，搜索到结点 0 的最短路径。另外，每扩展出一个结点，都记录下结点前驱以及引起这次状态转移的数字，方便最终输出结果。另外本题的 BFS 过程使用结点前驱判重，如果已经有前驱结点，则不再继续搜索。

举例来说，考虑 $n = 14$，$m = 3$，3 个禁用的数字分别是 7、8 和 4。为方便讨论起见，加入一个虚拟的根结点 x，表示根结点，则一开始附加所有可用的数字（1,2,3,5,6,9，参见图 2.66 中边上的数字）。一开始就建立从 x 到每个数字对应的余数结点的所有边。然后每个余数结点再添加一位进行状态转移，直到搜索出从 x 到 0 结点的最短路径。最短路径上每条边上的数字拼接起来就是所求的数字。

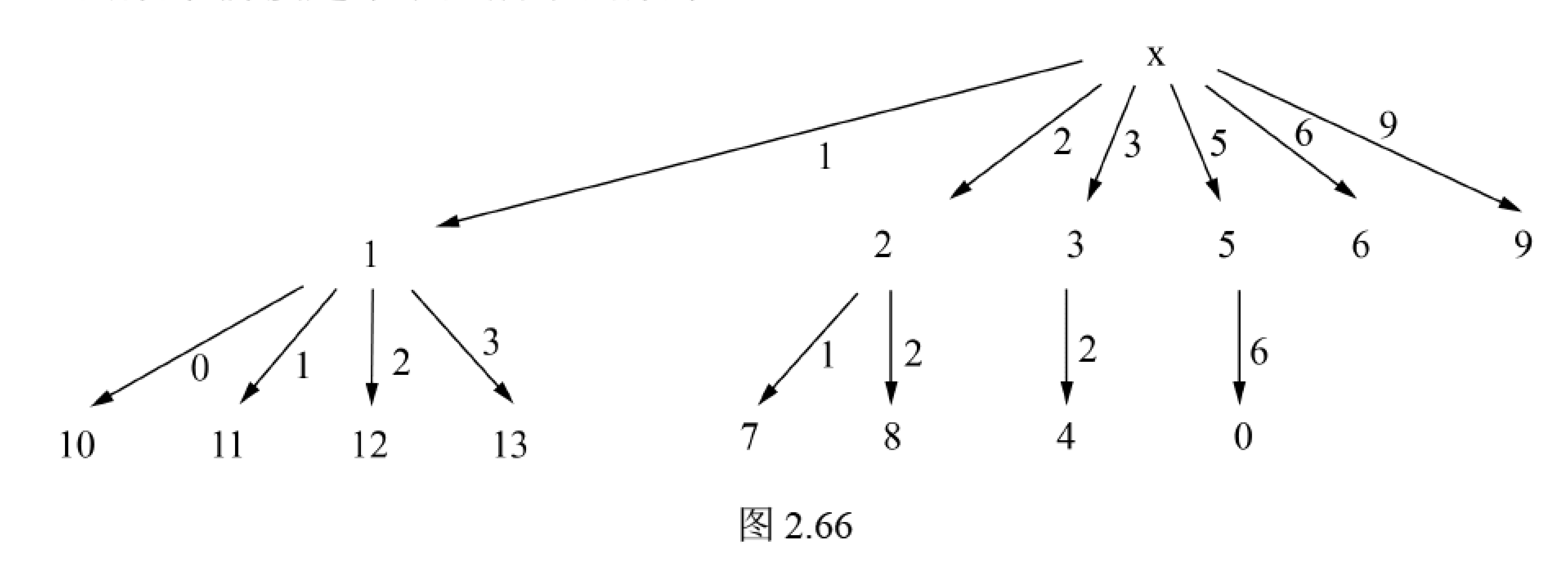


图 2.66

算法的时间复杂度为 $O(n)$，完整程序（C++11）如下：

```
const int N = 10000 + 4;
using namespace std;

int n, m, dig[16], pre[N], val[N]; //结点前驱，结点数字值
void solve() {
    queue<int> q;
    _rep(i, 1, 9) {
        if (dig[i]) continue;
        int mod = i%n;
        if (i >= n && mod == 0) { printf("%d\n", i); return; }
        if (pre[mod] != -1) continue;
        pre[mod] = 0, val[mod] = i, q.push(i);
    }
```

```
    function<void(int)> out = [&out](int x){
        if (pre[x]) out(pre[x]);
        printf("%d", val[x]);
    };

    while (!q.empty()){
        int mod = q.front();
        q.pop();
        _rep(i, 0, 9){
            if (dig[i]) continue;
            int nmod = (mod*10 + i)%n;
            if (nmod%n == 0){
                out(mod); printf("%d\n", i);
                return;
            }
            if (pre[nmod] != -1) continue;
            pre[nmod] = mod, val[nmod] = i, q.push(nmod);
        }
    }
    puts("-1");
}

int main(){
    for (int t=1; scanf("%d%d",&n,&m)==2; t++){
        printf("Case %d: ", t);
        fill_n(dig, 16, 0), fill_n(pre, N, -1);
        while (m--) dig[readint()] = 1;
        solve();
    }
    return 0;
}
```

习题 10-38　正多边形（Regular Polygon, UVa10824）

给出圆周上的 n（$n \leqslant 2000$）个点，选出其中的若干个组成一个正多边形，有多少种方法？输出每行包含两个整数 S 和 F，表示有 F 种选法得到正 S 边形。各行应按 S 从小到大排序。

【分析】

一个圆上的正 K 边形的顶点把圆分成 K 个角度相等的弧。从其中一个顶点就可以得到其他所有的顶点。

回到本题，输入时将每个点转换为弧度，然后排序。之后遍历所有的多边形顶点数 F，尝试从每个点出发构造一个正 F 边形，构造成功就记录下来。注意任何一个点都记录一下

曾经判断过的以其为顶点的多边形的顶点数，避免重复判断。算法的时间复杂度为 $O(n^3*\log(n))$。完整程序（C++11）如下：

```
using namespace std;
typedef long long LL;
const double EPS = 1e-8, PI = acos(-1);
const int MAXN = 2000 + 4;
int Vis[MAXN], Ans[MAXN];

int main(){
    LL N; double ang, x, y;
    vector<double> angs;
    auto dcmp = [](double a, double b){ return a - b < -EPS; };
    auto rg = [](double a){ if(a > 2*PI) a -= 2*PI; return a; };

    for(int t = 1; scanf("%lld", &N) == 1 && N; t++){
        angs.clear(), fill_n(Ans, N+1, 0LL), fill_n(Vis, N+2, 0);
        _for(i, 0, N) {
            scanf("%lf%lf", &x, &y);
            ang = atan2(y, x);
            if(ang < 0) ang += 2*PI;
            angs.push_back(ang);
        }
        sort(begin(angs), end(angs));
        _rep(K, 3, N) _for(i, 0, N){
            int pt = 1;
            if(Vis[i] == K) continue; //已经判断过包含 i 作为顶点的 K 边形
            Vis[i] = K;
            _for(p, 1, K){ //依次寻找 K-1 个顶点
                auto ppr = equal_range(begin(angs), end(angs),
                    rg(angs[i] + PI*2*p / K), dcmp);
                if(ppr.first == ppr.second) break;
                Vis[ppr.first-begin(angs)] = K;
                pt++;
            }
            if(pt == K) Ans[K]++;
        }

        printf("Case %d:\n", t);
        _rep(K, 3, N) if(Ans[K]) printf("%d %d\n", K, Ans[K]);
    }
}
```

习题 10-39　圆周上的三角形（Circum Triangle, UVa11186）

在一个圆周上有 n（$n\leqslant 500$）个点。不难证明，其中任意 3 个点都不共线，因此都可以组成一个三角形。求这些三角形的面积之和。

【分析】

三角形的面积都可以通过圆的面积减去 3 个弓形（图 2.67 左边的圆中的灰色部分）的面积得出。那么所有三角形的面积之和就可以如下计算：C_n^3 个圆面积 – 所有的三角形对应的弓形面积之和。

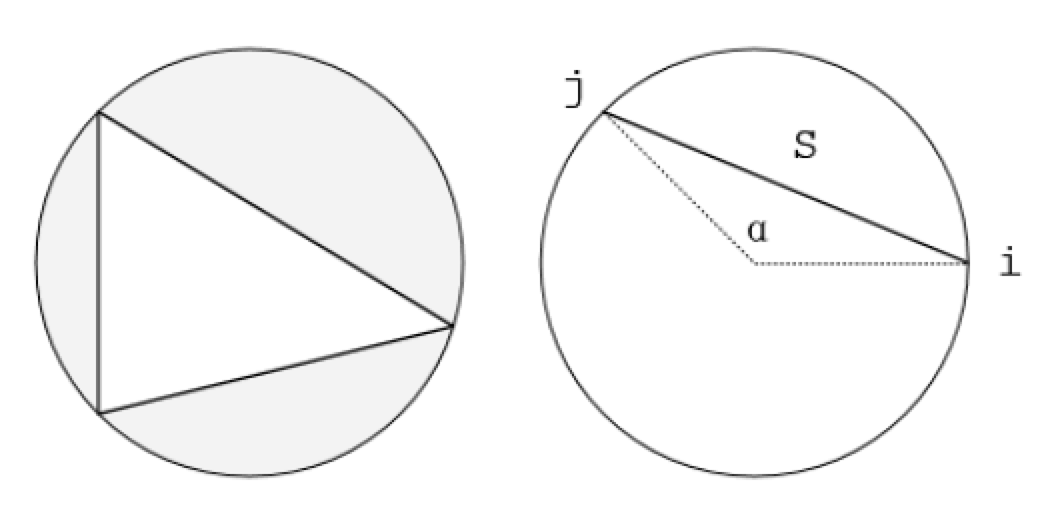


图 2.67

首先对于所有输入的点，将其转换为弧度，然后进行递增排序存到一个数组 A[n]中。而对于每两个弧度 i 和 j（$i<j$ 且 $A_i<A_j$）来说，i 到 j 之间会有 $j-i-1$ 个点，也就是说，会有 $j-i$ 个顶点在 ij 上方，并且以弦 ij 为底的三角形。也就是说，弦 ij 下方的弓形面积会在上述公式中出现 $j-i-1$ 次。而同理可知弦 ij 上方的弓形会出现 $n-2-(j-i-1)$次。记 $a=A_j-A_i$，则上方的弓形面积是 $S=\frac{aR^2}{2}-\frac{R^2\sin a}{2}=\frac{(a-\sin a)R^2}{2}$。下方的弓形面积就是 πR^2-S。二者的面积分别乘以出现次数再加起来就是：

$$S(n-2-(j-i-1))+(j-i-1)(\pi R^2-S)=S(n-2j+2i)+(j-i-1)\pi R^2$$

首先令 sum = $C_n^3 *\pi R^2$，遍历每一对 $i<j$，然后在 sum 上减去上述结果即可。注意当 $n<3$ 时，sum=0。而且可以先当作单位圆计算，令 $R=1$，输出时再乘以 R^2 即可。时间复杂度为 $O(n^2)$。完整程序（C++11）如下：

```
using namespace std;
const double PI = 2 * acos(0);
const int MAXN = 500 + 4;
int N, R;
double A[MAXN];

int main(){
    while(scanf("%d%d", &N, &R) == 2 && N && R){
        double ang, sum = 0;
        _for(i, 0, N) scanf("%lf", &ang), A[i] = ang/180*PI;
        sort(A, A + N);
        sum = N*(N-1)*(N-2)/6 * PI;
        _for(i, 0, N) _for(j, i+1, N){
```



```
            double a = A[j]-A[i], s = (a-sin(a))/2;
            sum -= s*(N-2*j+2*i) + (j-i-1)*PI;
        };
        if(N < 3) sum = 0;
        printf("%.0lf\n", round(sum*R*R));
    }
}
```

习题 10-40　实验法计算概率（Probability Through Experiments, ACM/ICPC Hatyai 2012, UVa12535）

输入圆的半径和圆上 n（n≤20000）个点的极角，任选 3 点能组成多少个锐角三角形？

【分析】

如图 2.68 所示，圆上的三角形，按顺时针方向记其顶点为 A、B、C。如果是钝角三角形，则圆周上 A 到 C 的角 AOC<180°。如果是直角三角形，则 AOC=180°，B 在 A、C 之间。如果是锐角三角形，则 AOC>180°。显然判断锐角三角形更麻烦些，所以可以使用排除法，求出所有三角形的个数减去直角和钝角三角形的个数即可。

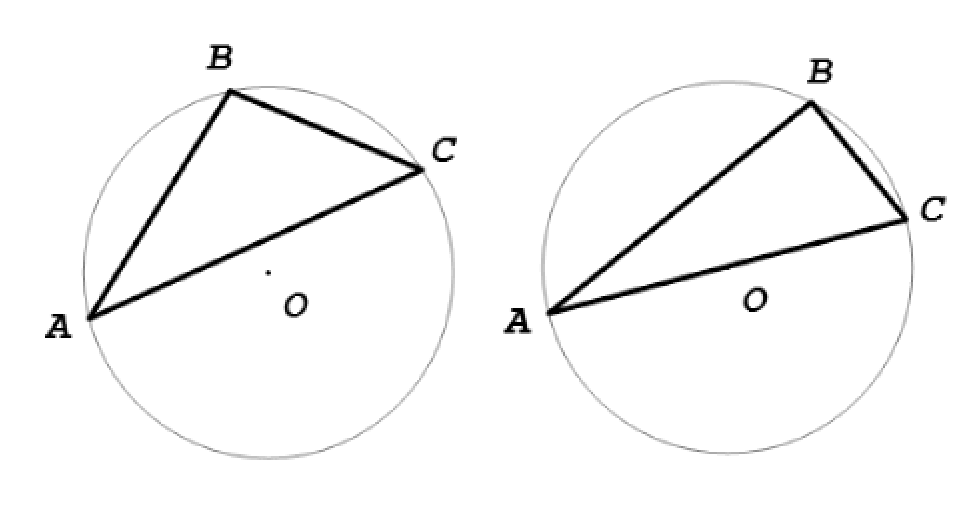


图 2.68

输入之后按弧度θ排序，为了方便处理，点集中要加入所有的θ+360。遍历所有的极角 A，然后查找所有的符合 A<B<C≤A+180 的 B 和 C 的个数。如果 A 到 A+180 中有 M 个点，则在总的三角形个数 $N(N-1)(N-2)/6$ 中减去 $M(M-1)/2$。完整程序（C++11）如下：

```
using namespace std;
typedef long long LL;
const double EPS = 1e-6;

int main(){
    LL N, R; double ang;
    vector<double> angs;
    for(int t = 1; scanf("%lld%lld", &N, &R) == 2 && N && R; t++){
        LL ans = N*(N-1)*(N-2)/6; angs.clear();
        _for(i, 0, N) {
            scanf("%lf", &ang);
            angs.push_back(ang);
            angs.push_back(ang + 360);
```

```
        }
        sort(begin(angs), end(angs));
        _for(ai, 0, N){
            auto ait = angs.begin() + ai;
            LL M = distance(ait + 1, //B,C 可选的点数
                upper_bound(ait, angs.end(), *ait+180, [](double d1, double
d2){
                    return d1 - d2 < -EPS;
                }));
            ans -= M*(M-1)/2;
        }

        printf("Case %d: %lld\n", t, ans);
    }
}
```

习题 10-42　网格中的三角形（Triangles in the Grid, UVa12508）

一个 n 行 m 列的网格有 n+1 条横线和 m+1 条竖线。任选 3 个点，可以组成很多三角形。其中有多少个三角形的面积位于闭区间[A,B]内？$1 \leq n, m \leq 200$，$0 \leq A < B \leq nm$。

【分析】

直接枚举三角形会非常麻烦，但是考虑到三角形的顶点都在横线和竖线的交点上，不难想到每个这种三角形都有一个最小包围矩形。那么首先是按照长宽来遍历这种矩形的个数，考虑矩形的左上角坐标以及长宽即可。因为三角形的面积在计算时都要除以 2，所以可以事先把 A 和 B 乘以 2，然后在计算三角形面积时就不除了，同时也避免计算误差。

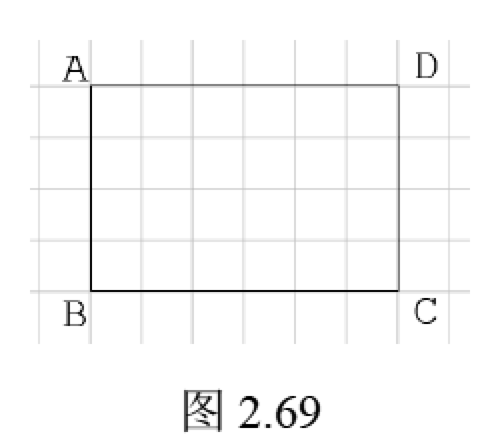


图 2.69

对于一个长宽为(r,c)的矩形 cell 来说，不妨对如图 2.69 所示的矩形的顶点做如下标记：

其包围的三角形可以分为以下几种情况，如图 2.70 所示。

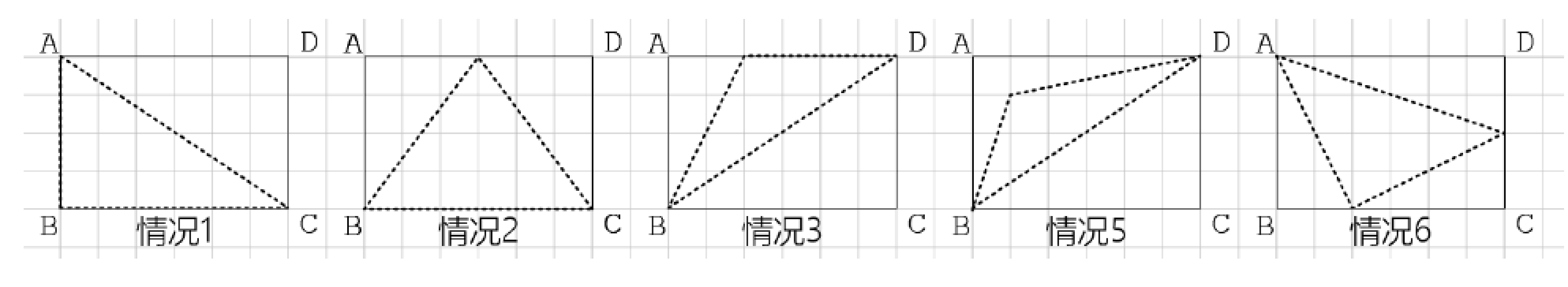


图 2.70

（1）3 个顶点都在 cell 顶点上，共 4 种，面积都是 $r*c$。

（2）只有两个顶点在 cell 顶点上，这两个顶点相邻，则第三个顶点在对边上。对第三个顶点进行计数可以得出这种三角形的个数为 $2*(r-1) + 2*(c-1)$。

（3）有两个顶点形成对角线，另外的顶点在水平边上。不妨设水平的那条边长度为 i，则 i 就需要满足：$A \leq r*i \leq B$ 且 $1 \leq i < c - 1$ -> $r \leq r*i < r*(c-1)$，也就是说 $\max(A,r) \leq r*i \leq \min(B,r*c-r)$。符合这个不等式的 i 的个数乘以 4 就是这种情况下的三角形个数。

（4）跟上述情况类似，有两个顶点形成对角线，另外的顶点在垂直边上。不妨设垂直的那条边长度为 i，则这种情况下三角形的个数就是符合以下不等式的 i 的个数乘以 4：$\max(A,c) \leqslant c*i \leqslant \min(B, r*c-c)$。

（5）有两个顶点形成对角线，另外的顶点在矩形内部。令第三个顶点的坐标为(i,j)，意思是离边 AB 的距离为 i，离 AD 的距离为 j。那么遍历所有的 i=1～r−1，当 i 确定时 j 就需要满足：$r*c - B - \mathrm{col} * i \leqslant r*j \leqslant r*c - A - \mathrm{Col} * i$。就是要计算符合此条件的 j 的个数然后乘以 4（对称性考虑）。

（6）只有一个顶点在四角上，另外两个点肯定都在跟这个点不相邻的边上。不妨设这个顶点就是 A，则另外两个顶点一定在 CD 和 BC 上，不妨设在 CD 上的顶点离 D 的距离为 i，在 BC 上的顶点距离 B 为 j。则三角形的面积为：$2rc - (i*c + j*r + (r-i)*(c-j)) = r*c - i*j$，遍历所有的 $i = 1$～r−1，对于指定的 i, j 就要满足 $A \leqslant r*c - i*j \leqslant B \rightarrow r*c - B \leqslant i*j \leqslant r*c - A$ 以及 $1 \leqslant j \leqslant c-1 \rightarrow i \leqslant i*j \leqslant i*(c-1)$，那么符合 $\max(r*c - B, i) \leqslant i*j \leqslant \min(r*c-A, i*(c-1))$这个不等式的 j 的个数乘以 4 就是符合这种情况的三角形个数。

遍历所有的 r 和 c，大小为(r,c)的矩形个数就是$(n-r+1) * (m-c+1)$。对这些矩形分别进行三角形计数并且加起来就是最终所求的三角形个数。

需要注意的是，各种情况都牵涉求符合形如 $L \leqslant K*X \leqslant R$ 的不等式的 X 的个数，可以将这个过程提取出来复用。完整程序如下：

```
using namespace std;
typedef long long LL;
int n, m, A, B;

void update(LL& cnt, int area, int c) { if(A <= area && area <= B) cnt+=c; }

int solve(int left, int right, int k) { //求方程 left ≤ k*x ≤ right 的解的个数
   if (left > right) return 0;
   left = (int)ceil(left / (double)k);
   right = (int)floor(right / (double)k);
   return right - left + 1;
}

LL solve(int r, int c) { //计算 r*c 的方格中有多少符合条件的三角形
   LL cnt = 0;
   int area = r*c;
   //三顶点都在 cell 顶点上，共 4 种
   update(cnt, area, 4);
   //只有两个顶点在 cell 顶点，并且这两个顶点不是对角线，第三个顶点在对边上
   update(cnt, area, 2*(r-1) + 2*(c-1));
   //有两个顶点形成对角线，另外的顶点在水平边上
   cnt += 4 * solve(max(A,r), min(B,r*c-r), r);
```

```
    //有两个顶点形成对角线，另外的顶点在垂直边上
    cnt += 4 * solve(max(A,c), min(B, r*c-c), c);
    //有两个顶点形成对角线，另外的顶点在矩形内部
    _for(i, 1, r) cnt += 4 * solve(max(r, area-B-c*i), min(area-r, area-A-c*i), r);
    //只有一个顶点在四角上，另外两个点肯定都在跟这个点不相邻的边上
    _for(i, 1, c) cnt += 4 * solve(max(i, area-B), min(i*r-i, area-A), i);
    return cnt;
}

LL solve() {
    LL cnt = 0;
    _rep(r, 1, n) _rep(c, 1, m) //遍历三角形所在的矩形的长宽
        cnt += solve(r, c) * (n-r+1) * (m-c+1);
    return cnt;
}

int main(){
    int T; cin>>T;
    while(T--) {
        cin>>n>>m>>A>>B;
        A*=2, B*=2;
        cout<<solve()<<endl;
    }
    return 0;
}
```

习题 10-43 整数对（Pair of Integers, ACM/ICPC NEERC 2001, UVa1654）

考虑一个不含前导零的正整数 X，把它去掉一个数字以后得到另外一个数 Y。输入 $X+Y$ 的值 $N(1\leqslant N\leqslant 10^9)$，输出所有可能的等式 $X+Y=N$。例如 $N=34$ 有两个解：31+3=34；27+7=34。

【分析】

记 n 的十进制表示形式的长度为 L，不妨设从 X 右边数第 i（$0\leqslant i<L$）位删除数字 x（$0\leqslant x<10$）得到 Y（如图 2.71 所示），记 $X=a*10^{i+1}+x*10^i+b$，$b<10^i$，则 $Y=a*10^i+b$。$X+Y=n$，故有 $11*a*10^i+x*10^i+2*b=N$。

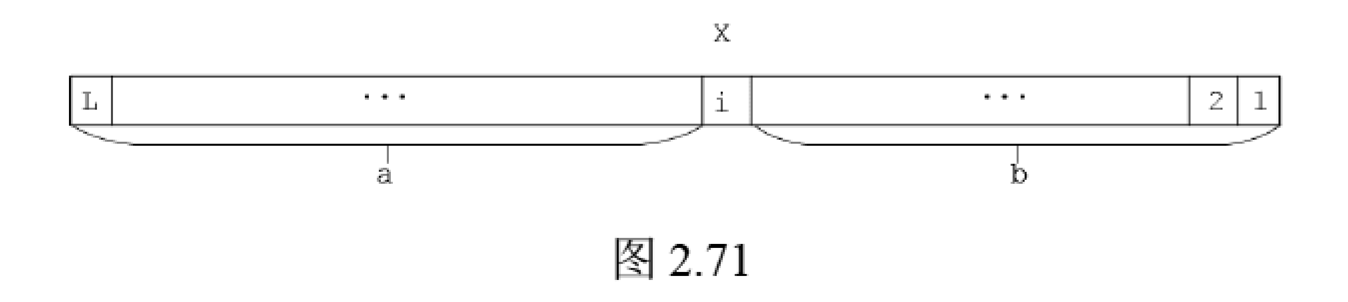


图 2.71

遍历所有的 i 和 x，固定 i 和 x 之后，求不定方程 $11*a*10^i+2*b=N-x*10^i$ 的所有满足 $a\geqslant 0$ 且 $0\leqslant b<10^i$ 整数解(a,b)，直接计算并输出 X 和 Y 即可。完整程序（C++11）如下：

```
using namespace std;
```

```
typedef long long LL;
typedef pair<LL, LL> PLL;

LL gcd(LL a, LL b, LL& x, LL& y){
    if(b==0){ x = 1, y = 0; return a; }
    LL res = gcd(b, a%b, y, x);
    y -= a/b*x;
    return res;
}

LL Pow10[15];
//求所有 a*x + b*y = n 的满足[0≤x, 0≤y<limit]的整数解
void solve(LL a, LL b, LL n, LL limit, vector<PLL>& res){
    res.clear();

    LL x,y,d = gcd(a,b,x,y), k = n/d; //求 ax + by = gcd(a,b) = d 的一组整数解
    if(n%d) return;                   //方程无解
    x *= k, y *= k, a /= d, b /= d;   //先求出一组解
    for(int i = 30; i >= 0; i--){     //将解规整到 y 离 a 最近
        LL t2 = 1LL<<i;
        if(y - t2*a >= 0) y -= t2*a, x += t2*b;
        if(y + t2*a < 0) y += t2*a, x -= t2*b;
    }
    if(y < 0) y += a, x -= b;
    while(x >= 0 && y < limit)
        res.push_back(make_pair(x, y)), y += a, x -= b;
}

void solve(){
    LL n; vector<PLL> tmp; map<LL, LL> ans;
    scanf("%lld", &n);
    auto llLen = [](LL x){
        int nLen = 0;
        while(x) nLen++, x /= 10;
        return nLen;
    };

    int nLen = llLen(n);
    _for(i, 0, nLen) _for(x, 0, 10){
        solve(Pow10[i+1]+Pow10[i], 2, n-x*Pow10[i], Pow10[i], tmp);
        for(auto p : tmp){
            LL X = p.first*Pow10[i+1] + x*Pow10[i] + p.second,
```

```
                Y = p.first*Pow10[i] + p.second;
            if(X == Y) continue;
            if(Y == 0){ if(X % 10) continue; } //X必须是10的2位数倍数才行
            ans[X] = Y;
        }
    }
    printf("%lu\n", ans.size());
    for(const auto &p : ans)
    printf("%lld + %0*lld = %lld\n", p.first, llLen(p.first)-1, p.second, n);
}

int main(){
    Pow10[0] = 1;
    _rep(i, 1, 12) Pow10[i] = Pow10[i-1]*10;
    int T; scanf("%d", &T);
    while(T--){
        solve();
        if(T) puts("");
    }
     return 0;
}
```

2.9 图论模型与算法

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 11 章。

习题 11-1 网页跳跃（Page Hopping, ACM/ICPC World Finals 2000, UVa821）

最近的研究表明，互联网上任何一个网页在平均情况下最多只需要点击 19 次就能到达任意一个其他网页。如果把网页看成一个有向图中结点，则该图中任意两点间最短距离的平均值为 19。

输入一个 n（$1\leqslant n\leqslant 100$）个点的有向图，假定任意两点之间都相互到达，求任意两点间最短距离的平均值。输入保证没有自环。

【分析】

模型就是带权有向图，权值是两点之间的距离，则使用 Floyd 算法就可以计算出任意两点之间的最短距离，最后求平均值。关于 Floyd 算法，请参考《算法竞赛入门经典（第 2 版）》中的 11.2.5 节。

习题 11-2 奶酪里的老鼠（Say Cheese, ACM/ICPC World Finals 2001, UVa1001）

无限大的奶酪里有 n（$0\leqslant n\leqslant 100$）个球形的洞。你的任务是帮助小老鼠 A 用最短的时间到达小老鼠 O 所在位置。奶酪里的移动速度为 10 秒一个单位，但是在洞里可以瞬间移动。



洞和洞可以相交。输入 n 个球的位置和半径，以及 A 和 O 的坐标，求最短时间。

【分析】

把起点 A 和目标点 O 都看作半径为 0 的球，则共有 n+2 个球。把每个球的球心看作结点，则两个球($p1$,$r1$)和($p2$,$r2$)所对应的结点之间的距离为 $d = |p1-p2| - r1 - r2$，如果 $d < 0$，说明两个球相交，可以认为 d=0。求出所有顶点之间的距离之后，任意两个球对应的结点之间都有一个距离为 d 的无向边。使用 Floyd 算法即可求出 A 和 O 之间的最短距离。

习题 11-3　因特网带宽（Internet Bandwidth, ACM/ICPC World Finals 2000, UVa820）

在因特网上，计算机是相互连通的，两台计算机之间可能有多条信息连通路径。流通容量是指两台计算机之间单位时间内信息的最大流量。不同路径上的信息流通是可以同时进行的。例如，图 2.72 中有 4 台计算机，总共 5 条路径，每条路径都标有流通容量。从计算机 1 到计算机 4 的流通总容量是 25，因为路径 1-2-4 的容量为 10，路径 1-3-4 的容量为 10，路径 1-2-3-4 的容量为 5。

请编写一个程序，在给出所有计算机之间的路径和路径容量后求出两个给定结点之间的流通总容量（假设路径是双向的，且两方向流动的容量相同）。

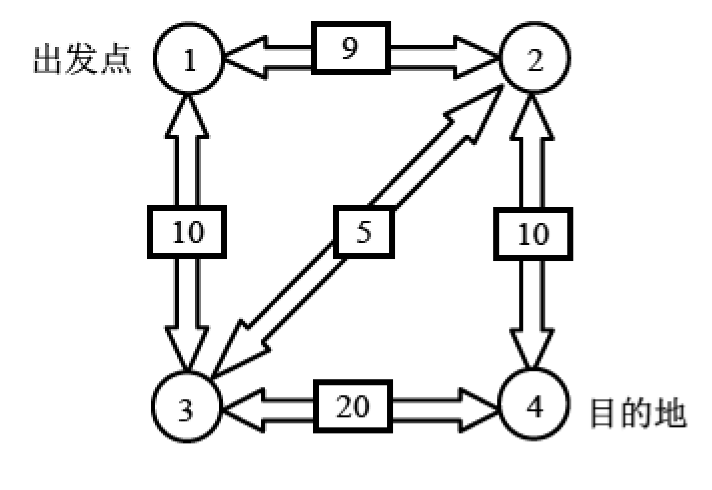


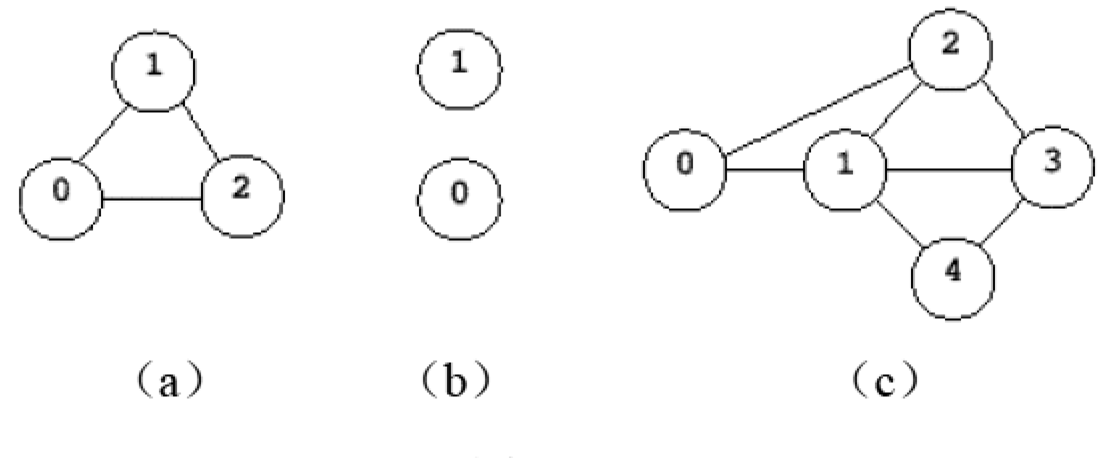
图 2.72

【分析】

把每个计算机看作一个结点，图 2.72 中每条路径对应有向图中两条边，容量都是路径的容量。之后应用 Dinic 算法求出给定结点之间的最大流即可。关于 Dinic 算法，请参考《算法竞赛入门经典——训练指南》中的 5.6.1 节。

习题 11-4　电视网络（Cable TV Network, ACM/ICPC SEERC 2004, UVa1660）

给定一个 n（n≤50）个点的无向图，求它的点连通度，即最少删除多少个点，使得图不连通。如图 2.73（a）所示的点连通度为 3，图 2.73（b）所示的连通度为 0，图 2.73（c）所示的点连通度为 2（删除 1 和 2 或者 1 和 3）。



（a）　（b）　（c）

图 2.73

【分析】

注意只要使得任意两个点不连通即可。将每个点 i 拆成两个点 i 和 $i+n$。建立一条有向边 $i \to i+n$，容量为 1。对于原图中的边(i,j)，建立两条边($i+n \to j$), ($j+n \to i$)，容量为 INF。两两遍历点对(u,v)，令 $u+n$ 为源点 s，v 为汇点 t。则要使 u 和 v 不连通所需删除的点对应于 s 到 t 之间由容量为 1 的边组成的割。这个点集大小的最小值，等于 $s \to t$ 最小割的容量。在所有的最小割容量中取最小值即是所求的点连通度。关于最小割的求法，请参考《算法竞

赛入门经典（第 2 版）》中的 11.4.3 节。完整程序如下：

```
using namespace std;

const int maxn = 50+1, maxm = maxn*maxn, INF = 1000000;
int n, m;
Edge edges[maxm];

Dinic<2*maxn, INF> d;

int solve(int u, int v) {
   d.ClearAll(2*n);
   _for(i, 0, n) d.AddEdge(i, i+n, 1);
   _for(i, 0, m) {
      const Edge& e = edges[i];
      d.AddEdge(e.from + n, e.to, INF);
      d.AddEdge(e.to + n, e.from, INF);
   }
   int s = u+n, t=v;
   return d.Maxflow(s, t);
}

int main(){
   char buf[4];
   while(scanf("%d%d", &n, &m) == 2) {
      int u,v;
      _for(i, 0, m) {
         Edge& e = edges[i];
         scanf(" (%d,%d)", &(e.from), &(e.to));
      }

      int ans = INF;
      _for(i, 0, n) _for(j, 0, n)
         if(i != j)
           ans = min(ans, solve(i,j));

      if(ans == INF) ans = n;
      printf("%d\n", ans);
   }
   return 0;
}
```



习题 11-5　方程（Equation, ACM/ICPC NEERC 2007, UVa1661）

输入一个后缀表达式 $f(x)$，解方程 $f(x)=0$。表达式包含四则运算符，且 x 最多出现一次。保证不会出现除以常数 0 的情况，即至少存在一个 x，使得 $f(x)$不会除 0。所谓后缀表达式，是指把运算符写在运算数的后面。例如，$(4x+2)/2$ 的后缀表达式为 4 x * 2 + 2 /。样例输入与输出如表 2.3 所示。

表 2.3

样 例 输 入	样 例 输 出
4 x * 2 + 2 /	x = −1/2
2 2 *	NONE
0 2 x / *	MULTIPLE

【分析】

第一步首先要将输入的表达式解析成表达式树，解析的过程实际上是递归的，输入运算符 op 的位置 end，得到 op 对应的表达式树，同时将 end 更新到解析出的表达式的左边位置。

解析完表达式之后，求解的过程依然是递归的，给定一个表达式树的根结点 p，以及这个表达式的值 v。因为题目条件中 x 只存在于一个结点，要么属于 p 的左子树，要么就是右子树。而 x 所在的子树的值就是未知的。首先根据 p 对应的运算符以及 v 求出 x 所在的子树的值（解一个一元方程），然后往下递归。发现任何无解或者有无数多个解时终止递归，主要是牵涉乘法和除法时有各种的特殊情况需要处理，详情请参见代码。

同时需要注意的是，求解的过程中的数值类型需要使用自定义的有理数类型，并且有理数的分子分母都需要使用 64 位的 long long 来保存，否则可能会溢出。

习题 11-6　括号（Brackets Removal, NEERC 2005, UVa1662）

给一个长度为 n 的表达式，包含字母、二元四则运算符和括号，要求去掉尽量多的括号。去括号规则如下：若 A 和 B 是表达式，则 A+(B)可变为 A+B，A−(B)可变为 A−B'，其中 B'为 B 把顶层“+”与“−”互换得到；若 A 和 B 为乘法项（term），则 A*(B)变为 A*B，A/(B)变为 A/B'，其中 B'为 B 把顶层“*”与“/”互换得到。本题只能用结合律，不能用交换律和分配律。

例如，((a−b)−(c−d)−(z*z*g/f)/(p*(t))*((y−u)))去掉括号以后为 a−b−c+d−z*z*g/f/p/t*(y−u)。

【分析】

首先使用《算法竞赛入门经典》中的 11.1.2 节中介绍的递归下降法对表达式进行解析，建立表达式树。树的结点包含：

（1）当前的运算符或字母。

（2）是否包含在括号内。

（3）运算符的优先级。

乘除的优先级比加减高。解析的过程中，如果发现一个序列左右两端有括号，则需要设置对应结点的标志位。

解析完成之后，对整棵树进行去括号的处理，处理顺序是从根结点递归往下：

（1）对于左结点，如果其优先级大于或等于当前结点，直接去掉其括号然后对其进行递归去括号处理。

（2）对于右结点，如果其优先级高于当前结点，直接去掉其括号即可。

（3）如果右结点优先级等于当前结点，并且发现需要当前结点为“-”或者“/”进行运算符的求反，如-(b−c)这样或者/(b/c)这样的表达式，那么就需要进行运算符取反处理，递归往下每次遇到左结点如果优先级等于上级结点，就取反。

（4）右结点取反之后，再对其进行去括号处理。

```
using namespace std;
const int MAXN = 1024;

/*MemPool 代码省略*/
struct Node {
   char ch;
   Node *left, *right;
   bool enclose;
   int opLevel;
   void init(char c){
      left = right = NULL; ch = c; enclose = false;
      opLevel = 0;
      if(ch == '*' || ch == '/') opLevel = 2;
      else if(ch == '+' || ch == '-') opLevel = 1;
   }
   const bool isOp() { return !islower(ch); }
};

string EX;
Node *pRoot;
MemPool<Node> nodePool;
Node* newNode(char c) {
   Node* ans = nodePool.createNew();
   ans->init(c);
   return ans;
}

ostream& operator<<(ostream& os, const Node* p) {
   if(!p) return os;
   if(p->enclose) os<<'(';
   os<<p->left<<p->ch<<p->right;
   if(p->enclose) os<<')';
   return os;
```

```
}

void reverse(Node* p) {
    assert(p);
    assert(p->isOp());
    char c = p->ch;
    switch(c) {
        case '+' : p->ch = '-'; break;
        case '-' : p->ch = '+'; break;
        case '*' : p->ch = '/'; break;
        case '/' : p->ch = '*'; break;
        default:
            assert(false);
    }

    Node *pl = p->left, *pr = p->right;
    if(pl && pl->isOp() && pl->opLevel == p->opLevel) reverse(pl);
    if(pr  &&  pr->isOp()  &&  !pr->enclose  &&  pr->opLevel  ==  p->opLevel)
reverse(pr);
}

void proc(Node* p) {
    //cout<<"proc "<<p<<endl;
    assert(p);
    if(!p->isOp()) return;
    Node *pl = p->left, *pr = p->right;
    if(pl && pl->isOp()){
        if(pl->opLevel >= p->opLevel) pl->enclose = false;
        proc(pl);
    }

    if(pr && pr->isOp()) {
        if(pr->opLevel > p->opLevel) pr->enclose = false;
        else if(pr->opLevel == p->opLevel) {
            if((p->ch=='/' || p->ch == '-') && pr->enclose) {
                pr->enclose = false;
                reverse(pr);
            }
            pr->enclose = false;
        }
        proc(pr);
    }
```

```
}

Node* parse(int l, int r) {
   assert(l <= r);
   char lc = EX[l], rc = EX[r];
   if(l == r) return newNode(lc);

   int p = 0, c1 = -1, c2 = -1;
   _rep(i, l, r) {
      switch(EX[i]) {
         case '(' : p++; break;
         case ')' : p--; break;
         case '+' : case '-' : if(!p) c1 = i; break;
         case '*' : case '/' : if(!p) c2 = i; break;
      }
   }

   if(c1 < 0) c1 = c2;
   if(c1 < 0) {
      Node* ans = parse(l+1, r-1);
      ans->enclose = true;
      return ans;
   }

   Node *ans = newNode(EX[c1]), *ln = ans->left = parse(l, c1-1),
      *rn = ans->right = parse(c1+1, r);
   assert(ans->opLevel);
   if(!ln->isOp()) ln->enclose = false;
   if(!rn->isOp()) rn->enclose = false;
   return ans;
}

int main(){
   while(cin>>EX) {
      nodePool.dispose();
      pRoot = parse(0, EX.size()-1);
      pRoot->enclose = false;
      proc(pRoot);
      cout<<pRoot<<endl;
   }
   return 0;
}
```



习题 11-7 电梯换乘（Lift Hopping, UVa 10801）

在一个假想的大楼里，有编号为 0～99 的 100 层楼，还有 n（$n \leqslant 5$）座电梯。你的任务是从第 0 楼到达第 k 楼。每个电梯都有一个运行速度，表示到达一个相邻楼层需要的时间（单位：秒）。由于每个电梯不一定每层都停靠，有时需要从一个电梯换到另一个电梯。换电梯时间总共 1 分钟，但前提是两座电梯都能停靠在换乘楼层。大楼里没有其他人和你抢电梯，但你不能使用楼梯（这是一个假想的大楼，你无须关心它是否真实存在）。

例如，有 3 个电梯，速度分别为 10、50、100，电梯 1 停靠 0、10、30、40 楼，电梯 2 停靠 0、20、30 楼，电梯 3 停靠第 0、20、50 楼，则从 0 楼到 50 楼至少需要 3920 秒，方法是坐电梯 1 到达 30 楼（300 秒），坐电梯 2 到达 20 楼（500 秒+换乘 60 秒），再坐电梯 3 到达 50 楼（3000 秒+换乘 60 秒），一共 300+500+60+3000+60=3920 秒。

【分析】

将每一层楼内每个电梯在该层的出口看作一个顶点，对于电梯 x 来说，假如它在第 i 层停靠，把 x 在 i 层的出口看作一个顶点记为 $x*100+i$。对于 x 的下一个停靠层 j 来说，$x*100+j$ 和 $x*100+i$ 可以连一条边，权值就是这个电梯从 i 到 j 层所需要的运行时间。而对于同样在 i 层停靠的每个电梯 y 来说，$x*100+i$ 到 $y*100+i$ 也连一条边，权值为 60（同层换乘）。

图建好之后，遍历所有的形为 $x*100$ 的点 u 和 $y*100+k$ 的 v，使用 dijkstra 算法求出所有 u、v 间最短距离的最小值即为所求结果。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 5, MAXK = 100, MAXP = MAXN*MAXK;
const int INF = 100000 + 10;

Dijkstra<MAXP+1> solver;
int n, k, T[MAXN];
vector<int> Level[MAXK];
char buf[512];
int readint() { int x; scanf("%d", &x); return x; }

int solve() {
   solver.init(n*MAXK+1);
   _for(i, 0, n) T[i] = readint();
   gets(buf);
   _for(e, 0, n){
      gets(buf);
      istringstream iss(string(buf), istringstream::in);
      bool first = true;
      int l1, l; //上一层，当前层
      while(iss>>l) {
         if(first) first = false;
         else {
```

```
                int v = MAXK*e + l1, v2= MAXK*e + l;
                int dist = (l-l1)*T[e];
                solver.addEdge(v,v2,dist);
                solver.addEdge(v2,v,dist);
                //printf("[%d,%d]-%d\n",v, v2, dist);
            }
            Level[l].push_back(e);
            l1 = l;
        }
    }

    _for(i, 0, MAXK) {
        vector<int>& li = Level[i];
        _for(j, 0, li.size()){
            _for(m, j+1, li.size()) {
                int e1 = li[j], e2 = li[m], v1 = e1*MAXK+i, v2 = e2*MAXK+i;
                solver.addEdge(v1,v2,60);
                solver.addEdge(v2,v1,60);
                //printf("[%d,%d]-%d\n",v1, v2, 60);
            }
        }
    }

    vector<int>& L0 = Level[0];
    vector<int>& Lk = Level[k];
    if(L0.empty() || Lk.empty()) return 0;

    int ans = INF;
    for(auto i : L0) {
        solver.dijkstra(i * 100);
        for(auto j : Lk) ans = min(ans, solver.d[j*100 + k]);
    }

    return ans;
}

int main(){
    while(scanf("%d %d\n", &n, &k) == 2) {
        int ans = solve();
        if(ans != INF) printf("%d\n", ans);
        else puts("IMPOSSIBLE");
    }
```



```
    return 0;
}
```

习题 11-8 净化器（Purifying Machine, ACM/ICPC Beijing 2005, UVa1663）

给 *m* 个长度为 *n* 的模板串。每个模板串包含字符 0、1 和最多一个星号“*”，其中星号可以匹配 0 或 1。例如，模板 01*可以匹配 010 和 011 两个串，而模板集合{*01, 100, 011}可以匹配串{001, 101, 100, 011}。

你的任务是改写这个模板集合，使得模板的个数最少。例如，上述模板集合{*01, 100, 011}可以改写成{0*1, 10*}，匹配到的字符串集合仍然是{001, 101, 100, 011}。$n \leqslant 10$，$m \leqslant 1000$。

【分析】

拿到一个输入串之后，首先展开成可以匹配的串集合 S，S 中的元素用对应的整数值来表示。例如{*01, 100, 011}→{101, 001, 100, 011}→{5, 1, 4, 3}。

两两判断这个 S 中的点 s1, s2，如果二者的二进制只有一位不同，在 s1 和 s2 之间连一条边，对应一个带“*”的模板串，其中“*”的位置就是 s1 和 s2 不同的那一位。而 s1 和 s2 中 1 的个数的奇偶性必然不同。可以按照这个奇偶性将所有点分成两类，就形成一个二分图。而这个二分图的每一个匹配都对应于一个带“*”的模板集合。求此二分图的最大匹配，假设其中有 *n* 条边，则所求的模板集合的最小值就是$|S|-n$。

完整程序如下：

```
using namespace std;

int countBit(int x, int w){
    int b = 1, ans = 0;
    _for(i, 0, w) ans += ((b&x)!=0), b <<= 1;
    return ans;
}

string printbin(int x, int w){
    string buf;
    for(int i = w-1; i >= 0; i--) buf += (((1<<i)&x) > 0 ? 1 : 0) + '0';
    return buf;
}

const int MAXM = 1024 + 5;
BPM<MAXM> solver;
string S[MAXM];
int N, M, Set[MAXM];
int main(){
    string buf;
```

```
set<int> vs; // verticles
while(true){
    cin>>N>>M;
    if(N == 0) break;
    int sz = 1<<N;
    fill_n(Set, sz, 0);
    solver.init(sz, sz);
    _for(i, 0, M){
        cin>>buf;
        int x = -1, v = 0, bit = 1;
        _for(j, 0, buf.size()) {
            char c = buf[j];
            if(c == '*') x = j, bit = 1;
            else bit = c - '0';
            v = v*2 + bit;
        }
        Set[v] = 1;
        if(x != -1) {
            v &= ~(1<<(N-x-1));
            Set[v] = 1;
        }
    }

    //左边是偶数个 1 的串，右边是奇数个 1 的模板串
    int cnt = 0;
    _for(i, 0, sz) {
        if(!Set[i]) continue;
        // cout<<printbin(i, N)<<", ";
        cnt++;
        if(countBit(i, N)%2 == 1) continue;

        int b = 1;
        _for(b, 0, N){
            int j = (1<<b)^i;
            if(Set[j]) solver.AddEdge(i, j);
        }
    }

    int m = solver.solve();
    cout<<cnt-solver.solve()<<endl;
}
```



```
    return 0;
}
```

习题 11-9 机器人警卫（Sentry Robots, ACM/ICPC SWERC 2012, UVa12549）

在一个 Y 行 X 列（$1 \leqslant Y, X \leqslant 100$）的网格里有空地（.）、重要位置（*）和障碍物（#），如图 2.74 所示。用最少的机器人看守所有重要位置。每个机器人要放在一个格子里，面朝上、下、左、右 4 个方向之一。机器人会发出激光，一直射到障碍物为止，沿途都是看守范围。机器人不会阻挡射线，但不同的机器人不能放在同一个格子。

```
  Grid           Solution
. * * . .      . → * . .
. * # * .      . ↑ # ↑ .
. # * . .      . # ↓ . .
. . * . .      . . * . .
```

图 2.74

【分析】

需要注意的是，机器人放到重要位置上，看守这个重要位置。

考虑没有障碍物的简单情况，实际上就是《算法竞赛入门经典——训练指南》的第 5.5.4 节中的例题 27（UVa11419）。建模过程如下：把每一行看作一个 X 结点，每一列看作一个 Y 结点，每个重要位置看作一条边连接相应的行结点和列结点。同一行或列只需要放置一个机器人，就可以看守所在同一行或同一列上的所有重要位置。这样就要求选择一组结点，使得每个所有重要位置对应的边都至少有 1 个结点被覆盖。这样就转换成为求二分图的最小覆盖，可以证明最小覆盖等于最大匹配数。

这样，问题的关键就在于能否把带有障碍物的模型转换成为不带障碍物的模型。首先按照行进行转换：从上到下，从左到右遍历每个点，遇到障碍物时，就将障碍物以及所有右边的点（包括障碍物和重要位置）向下平移一行，这样将左边的点和右边的点分割成上下两行。如图 2.75 所示为每个点的坐标以及做了上述水平转换后的坐标。这样，被障碍物隔开的两边的点就可以分成独立的两行，分别由单独机器人进行看守。同理，按行进行转换之后，继续按照列进行类似的转换。

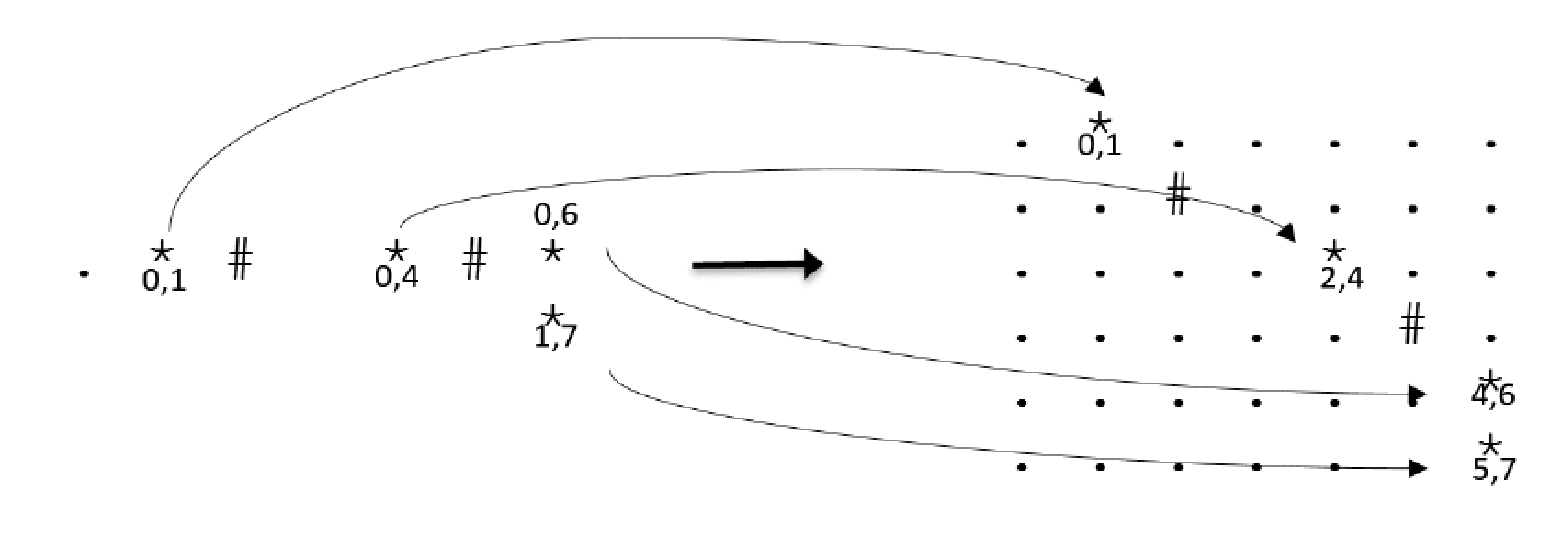


图 2.75

转换完成之后，点水平和垂直坐标的最大值分别是 X+2*W、Y+2*W，这也是建成的二分图的 X 和 Y 点数的最大值，而重要位置的个数（也就是说二分图的边数）依然是 P。完整程序（C++11）如下：

```
using namespace std;
const int MAXX = 100+5;

int Y, X, P, W;
vector<Point> points;

BPM<MAXX*MAXX*2> solver;

void dbgPrint() {
    cout<<X<<"*"<<Y<<endl;
    string line(X, '.');
    vector<string> M(Y, line);
    for(auto& p : points) {
        //cout<<"p-> "<<p.y<<", "<<p.x<<endl;
        assert(p.y < Y); assert(p.x < X);
        M[p.y][p.x] = p.ch;
    }
    _for(i, 0, Y) cout<<M[i]<<endl;}

int solve() {
    sort(points.begin(), points.end(), [](const Point& p1, const Point& p2){
      return p1.y < p2.y || (p1.y==p2.y && p1.x < p2.x); });
    int dy = 0;                    //垂直拆开
    for(auto& p : points) {
        bool isOb = (p.ch == '#');
        if(isOb) dy++;
        p.y += dy;
        if(isOb) dy++;
    }
    Y += dy;

    sort(points.begin(), points.end(), [](const Point& p1, const Point& p2){
      return p1.x < p2.x || (p1.x==p2.x && p1.y < p2.y);});
    int dx = 0;                    //水平拆开
    for(auto& p : points) {
        bool isOb = (p.ch == '#');
        if(isOb) dx++;
        p.x += dx;
        if(isOb) dx++;
```

```
    }
    X += dx;

    solver.init(X, Y);
    for(const auto& p : points)
      if(p.ch == '*') solver.AddEdge(p.x, p.y);
    vector<int> t1, t2;
    solver.mincover(t1, t2);
    return t1.size() + t2.size();
}

int main(){
    int C; cin>>C;
    _rep(t, 1, C){
        cin>>Y>>X>>P;
        points.clear();
        Point p;
        _for(i, 0, P){
            cin>>p.y>>p.x;
            p.ch = '*', p.x--, p.y--;
            points.push_back(p);
        }
        cin>>W;
        _for(i, 0, W) {
            cin>>p.y>>p.x;
            p.ch = '#', p.x--, p.y--;
            points.push_back(p);
        }

        int ans = solve();
        cout<<ans<<endl;
    }
    return 0;
}
```

习题 11-11 占领新区域（Conquer a New Region, ACM/ICPC Changchun 2012, UVa1664）

假设 n（$n \leqslant 200000$）个城市形成一棵树，每条边有权值 $C(i,j)$。任意两个点的容量 $S(i,j)$ 定义为 i 与 j 唯一通路上容量的最小值。找一个点（它将成为中心城市），使得它到其他所有点的容量之和最大。

【分析】

因为是一棵树，任意一条边都可以将整个图分成两棵树，所求的中心点一定是在其中

一棵树中，由此想到一开始可以从每个点开始求中心点，然后不断合并。

参考 Kruskal 算法的思路，首先初始化并查集，一开始每个集合的代表元素都是中心点，维护每个点集的元素个数 Cnt[i]以及中心点到其他点的容量之和 WS[i]，其中 i 是这个元素的代表元，一开始 i 肯定是所在点集的中心城市，且 WS[i]=0。

把所有边按权值从大到小排序，依次遍历每条边 e，使用 e 将已有两个点集连起来。记这两个点集为 A、B，中心点分别是 a、b。因为已经排序，e 一定是连接来自两个点集的边中权值最小的，而且分别来自 A 和 B 的任意两点之间的唯一通路一定经过 e，所以通路的容量一定为 e 的权值。

然后考虑 A 和 B 合并之后的点集，如果要把 b 作为其中心点，产生的新的点集的容量就是 W_b=WS[b]+Cnt[a]*w，其中 w 是 e 的权值。而 b 到 A 中每个点的容量都是 w。反过来把 a 作为中心点产生的新点集容量是 W_a=WS[a]+Cnt[b]*w。不妨设 $W_a>W_b$，把 B 合并到 A 中，*a* 就是新的点集的符合要求的中心点。所有边遍历完成之后，合并完成的集合的代表元就是所求的中心点，算法的时间复杂度为 O(N)。完整程序如下：

```
using namespace std;
const int MAXN = 200000 + 4;
typedef long long LL;
struct Edge{
   int from, to, weight;
   bool operator<(const Edge& rhs) const { return weight > rhs.weight; }
};

Edge edges[MAXN];
LL WS[MAXN], Pa[MAXN], Cnt[MAXN]; //WS[i]表示以 i 为根节点的树的边权和
int main(){
   int N;
   while(scanf("%d", &N) == 1 && N){
      _for(i, 1, N) {
         Edge& e = edges[i];
         scanf("%d%d%d", &(e.from), &(e.to), &(e.weight));
      }

      sort(edges + 1, edges + N);
      _rep(i, 1, N) Cnt[i] = 1, Pa[i] = i, WS[i] = 0;
      function<int(int)> find_pa = [&find_pa](int i){
         return Pa[i] == i ? i : (Pa[i] = find_pa(Pa[i]));
      };

      auto merge = [](int from, int to, LL v){ //合并，更新容量和点集大小
         Pa[from] = to, Cnt[to] += Cnt[from], WS[to] = v;
      };
```

```
        LL ans = 0;
        _for(i, 1, N){
            const Edge& e = edges[i];
            int a = find_pa(e.from), b = find_pa(e.to);
            LL wb = WS[b] + Cnt[a]*e.weight, wa = WS[a] + Cnt[b]*e.weight;
            if(wb > wa) merge(a, b, wb); else merge(b, a, wa);
            ans = max(wa, wb);
        }

        printf("%lld\n", ans);
    }
    return 0;
}
```

本题的严格证明留给读者思考。

习题 11-12　岛屿（Islands, ACM/ICPC CERC 2009, UVa1665）

输入一个 $n*m$（$1\leqslant n,m\leqslant 1000$）矩阵，每个格子里都有一个$[1,10^9]$正整数。再输入 T（$1\leqslant T\leqslant 10^5$）个整数 t_i（$0\leqslant t_1\leqslant t_2\leqslant\cdots\leqslant t_T\leqslant 10^9$），对于每个 t_i，输出大于 t_i 的正整数组成多少个四连块。如图 2.76 所示，大于 1 的正整数组成两块，大于 2 的组成 3 块。

1	2	3	3	1
1	3	2	2	1
2	1	3	4	3
1	2	2	2	2

1	2	3	3	1
1	3	2	2	1
2	1	3	4	3
1	2	2	2	2

图 2.76

评论：这个题目虽然和图论没什么关系，但是可以用到本章介绍的某个数据结构。

【分析】

因为牵涉集合的查询合并，首先可以使用并查集来表示连续的格子组成的集合。最简单的做法就是从大到小遍历所有的 t，每次查找对应的格子，但是粗略估计时间复杂度至少是 $T*n*m=10^{11}$，必然会超时。这样可以将所有的格子按照值从大到小排序，然后每次只查找对应的格子，并且复用上次遍历的结果。

具体来说，可以用一个结构 P$\{x,y,v\}$表示位置是$[x,y]$且值为 v 的数字。然后按照 v 从大到小对所有的 P 排序，初始每个 P 属于一个独立的集合（块）。之后按照从大到小的顺序，遍历 t_i。记 ans 为开始遍历时符合条件的集合的个数。$i = T$ 则 ans = 0，否则 ans 初始就是 t_{i+1} 对应的结果。从大到小依次扫描 $t_{i+1}>v>t_i$ 的所有点 p。每遍历到一个 p，ans 加 1，然后依次查看 p 在矩阵中的 4 个邻居 pn，如果 pn.v > t_i，且 pn 和 p 不在同一个集合，则将 pn 和 p 所在的集合合并，ans 减 1。扫描完符合条件的 P 之后 ans 就是符合 $v > t_i$ 的集合个数。

虽然是两层循环，但是内层循序实际上是不重复地遍历所有格子，总的循环次数是 nm。

时间复杂度为 $O(nm*\alpha(mn)+T)$。这里 α 表示并查集查找过程的时间复杂度，基本上可以认为是常量，不大于 4，具体的分析请参考《算法导论》一书的 21.4 节。完整程序（C++11）如下：

```
using namespace std;
inline int readint() { int x; scanf("%d", &x); return x; }

struct Point{
   int x,y,v;
   void init(int r, int c, int value){
      assert(r >= 0), assert(c >= 0);
      x = r, y = c, v = value;
   }
};

const int MAXM = 1024, MAXN = 1024, MAXT = 100000 + 5;
int m, n, T, t[MAXT], Ans[MAXT], indice[MAXN][MAXM], pa[MAXN*MAXM];
Point points[MAXN*MAXM];
vector<int> tmp; //储存 p 的邻居
int findPa(int i) { return pa[i] == i ? i : (pa[i] = findPa(pa[i])); }
void getAdjs(const Point& p, vector<int>& ans) {
   ans.clear();
   int r = p.x-1, c = p.y;
   if(r>=0) ans.push_back(indice[r][c]);
   r=p.x+1;
   if(r<n) ans.push_back(indice[r][c]);
   r = p.x; c = p.y-1;
   if(c>=0) ans.push_back(indice[r][c]);
   c = p.y+1;
   if(c<m) ans.push_back(indice[r][c]);
}

void solve() {
   scanf("%d%d", &n, &m);
   int psz = 0;
   _for(i, 0, n) _for(j, 0, m) points[psz++].init(i,j,readint());
   sort(points, points+psz, [](const Point& p1, const Point& p2){
      return p1.v > p2.v;
   });

   _for(i, 0, psz){
      const Point& p = points[i];
      pa[i] = i;
      indice[p.x][p.y] = i;
```

```
    }

    T = readint();
    _for(i, 0, T)  t[i] = readint();

    int pi = 0;
    Ans[T] = 0;
    for(int i = T-1; i>=0; i--) {
        int& ans = Ans[i];
        ans = Ans[i+1];
        while(pi<psz && points[pi].v>t[i]) {
            ans++;
            getAdjs(points[pi], tmp); //遍历 p 的上下左右 4 个邻居
            for(auto j : tmp){
                Point& pn = points[j];
                if(pn.v <= t[i]) continue;
                int pnpa = findPa(indice[pn.x][pn.y]);
                if (pnpa == findPa(pi)) continue;
                pa[pnpa] = pi;
                ans--;
            }
            pi++;
        }
    }
}

int main(){
    int Z = readint();
    _for(z, 0, Z) {
        solve();
        _for(i, 0, T) printf("%d ", Ans[i]);
        puts("");
    }
    return 0;
}
```

习题 11-14 乱糟糟的网络（Network Mess, ACM/ICPC Tokyo 2005, UVa1667）

有一棵 n（$n\leqslant 50$）个叶子的无权树。输入两两叶子的距离，恢复出这棵树并输出每个非叶子结点的度数。

【分析】

面对这种无根的树型结构，一般一开始就任意取一个叶子结点 u 作为根。记 $f(i,j)$为 i 和 j 两个叶子之间的距离。

递归处理：输入以 u 为根的子树的所有叶子结点以及 u 到所有叶子的距离。枚举它的任意两个叶子 i、j，如图 2.77 所示。

（1）$f(u,i)+f(u,j)>f(i,j)$，说明 i 和 j 是在 u 为根的树的同一个子树。

（2）$f(u,i)+f(u,j)=f(i,j)$，说明 i 和 j 不在同一棵子树中。

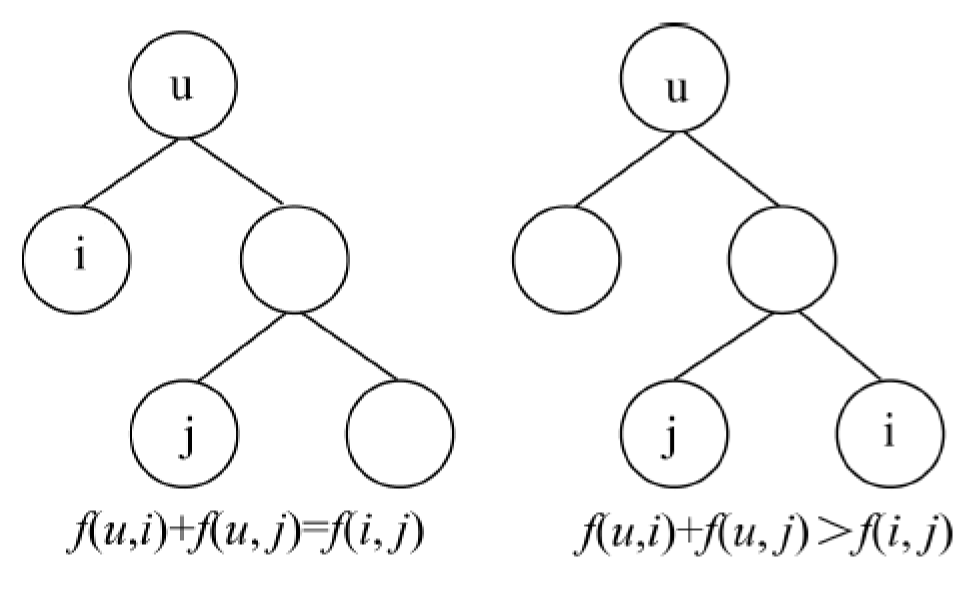


图 2.77

然后用并查集对所有的 u 的叶子进行分组，分成不同的子树，同时创建对应的子树根。在往下递归之前，应该将所有的叶子结点到根结点的距离都减一。

在这个过程中，对于 $f(u,i)=1$ 的情况，说明 i 就是 u 的直接叶子结点，不需要往下递归，但是需要构造一个相应的叶子结点方便进行统计。通过上述逻辑，就可以找出哪些叶子在同一棵子树中，以及 u 有多少棵子树。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 64;
struct Node{ vector<Node*> nodes; };
typedef Node* PNode;
MemPool<Node> nodePool;
vector<int> M[MAXN];
int N, pa[MAXN];
int getPa(int i) { return i == pa[i] ? i : (pa[i] = getPa(pa[i])); }

//建树，叶子结点，点到根结点的距离，根结点指针，父结点，度数数组
void buildTree(const vector<int>& leaves, vector<int>& dist, PNode root,
PNode fa, vector<int>& degs) {
    memset(pa, 0, sizeof(pa));
    for(const auto l : leaves) pa[l] = l;

    int isLeaf[MAXN];        //直接的叶子
    memset(isLeaf, 0, sizeof(isLeaf));

    for(auto lit = begin(leaves); lit != end(leaves); lit++) {
        int li = *lit;
        if(dist[li] == 1) {  //到根结点距离为 1，就是直接的叶子结点
            root->nodes.push_back(nodePool.createNew());
```

```
            isLeaf[li] = 1;
            continue;
        }
        for(auto rit = lit+1; rit != end(leaves); rit++){
            int ri = *rit;
            if(dist[ri] == 1) continue;
            if(dist[li] + dist[ri] > M[li][ri]) pa[ri] = li; //相同的子树
        }
    }

    for(auto i : leaves) dist[i]--;
    map<int, vector<int> > subTrees;                     //每颗子树
    for(auto i : leaves) if(!isLeaf[i]) subTrees[getPa(i)].push_back(i);

    for(const auto& p : subTrees){
        root->nodes.push_back(nodePool.createNew());
        buildTree(p.second, dist, root->nodes.back(), root, degs);
    }

    if(fa && !root->nodes.empty()) degs.push_back(root->nodes.size() + 1);
}

int main(){
    while(scanf("%d", &N) == 1 && N) {
        _for(i, 0, N){
            M[i].clear();
            _for(j, 0, N) M[i].push_back(readint());
        }
        vector<int> leaves, dist(M[0].begin(), M[0].end()), degs;
        _for(i, 1, N) leaves.push_back(i);

        buildTree(leaves, dist, nodePool.createNew(), NULL, degs);
        sort(degs.begin(), degs.end());
        bool first = true;
        for(auto d : degs){
            if(first) first = false; else printf(" ");
            printf("%d", d);
        }
        puts("");
        nodePool.dispose();
    }
    return 0;
```

```
}
```

习题 11-16　交换房子（Holiday's Accomodation, ACM/ICPC Chengdu 2011, UVa1669）

有一棵 n（$2 \leqslant n \leqslant 10^5$）个结点的树，每个结点住着一个人。这些人想交换房子（即每个人都要去另外一个人的房子，并且不同人不能去同一个房子）。要求安排每个人的行程，使得所有人旅行的路程长度之和最大。

【分析】

因为是一棵树，安排好的符合条件的行程中，假如有两对结点（u1,u2）和（v1,v2）。u1 要到 v1，u2 要到 v2，而且这两条路径不相交。那么在两条路径上分别选择点 p1 和 p2，则重新做如下安排：u1→p1→p2→v2，u2→p2→p2→v2，其中记 p1→p2 的路径长度为 pl，显然这样安排的路径相交而且长度之和比之前的长度长 2*pl。由此可以证明，任意两个行程的路径必定相交。

首先任选一个结点作为根，构造整棵树，同时对每个结点 u 记录以 u 为根结点的子树的结点个数 D_u。然后对于每个长度为 w 的边 e，假设这条边将树分成 A、B 两颗子树，根据上述结论可以得出经过这条边的路径的条数就是 $m_e = 2*\min\{D_A, n-D_A\}$。对所有边求 $\sum w_e*m_e$ 即可得所求结果。至于如何求每颗子树的结点个数，可以使用《算法竞赛入门经典（第 2 版）》中的 9.4.2 节，《树的重心》部分的树形 DP 过程。

习题 11-17　王国的道路图（Kingdom Roadmap, ACM/ICPC NEERC 2011, UVa1670）

输入一个 n（$n \leqslant 100000$）个结点的树，添加尽量少的边，使得任意删除一条边之后图仍然连通。如图 2.78 所示，最优方案用虚线表示。

【分析】

要满足所求的条件，保证结果的图中没有桥即可，可将每个叶子都和其他叶子相连。任意选择一个度数为 1 的非叶子结点作为树根，然后依次对每颗子树进行递归操作。

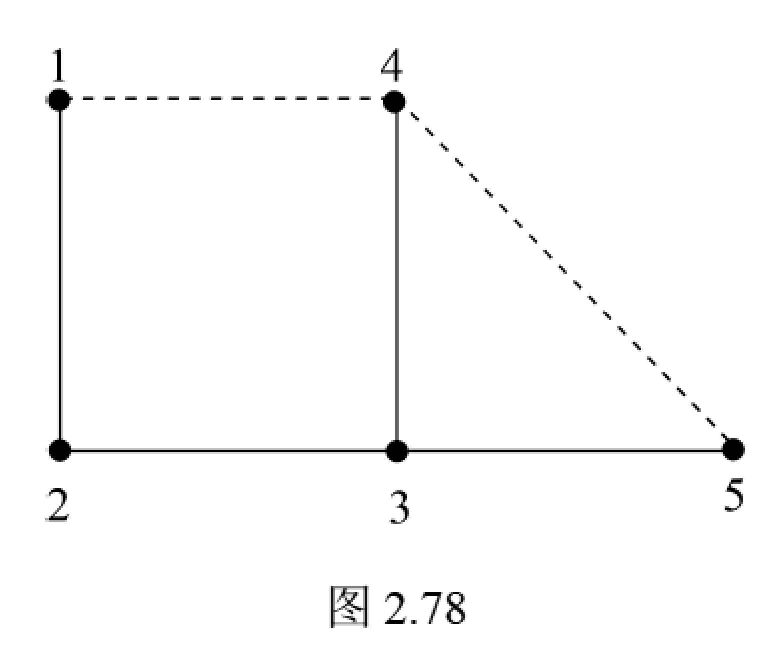


图 2.78

对于每颗子树：把悬在桥上的叶子依次配对连起来，但是要保证留下至少一个传回给父结点，以便最终和根结点配对。对于根结点，如果剩下两个叶子，就把它们连起来，如果剩一个，就在这个叶子和根结点之间连一条边。

证明留给读者思考。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 100000 + 4;
struct Edge { int from, to; };
struct Node {
   int id, deg, vis;
   vector<int> adj; //children;
   void init(int i) { id = i, deg = 0, adj.clear(); }
};
int n, root;
```



```
Node nodes[MAXN];
//去掉u为结点的子树中的桥
void connect(int u, int pa, vector<int>& ls, vector<Edge>& edges) {
    const Node& nu = nodes[u];
    ls.clear();
    if(nu.deg == 1) { ls.push_back(u); return; }

    vector<int> lvs;              //u的子结点的遗留结点
    for(auto v : nu.adj) {
        if(v == pa) continue;
        lvs.clear();
        connect(v, u, lvs, edges);
        if(ls.size() + lvs.size() > 2){
            assert(!ls.empty() && !lvs.empty());
            edges.push_back(Edge{ls.back(), lvs.back()});
            ls.pop_back(), lvs.pop_back();
        }
        for(auto& lv : lvs) ls.push_back(lv);
    }
}

void solve(vector<Edge>& edges) {
    edges.clear();
    vector<int> ls;               //悬着的点
    connect(root, 0, ls, edges);
    assert(ls.size() <= 2);

    if(ls.size() == 2) edges.push_back(Edge{ls[0], ls[1]});
    else edges.push_back(Edge{root, ls[0]});
}

int main(){
    int u,v;
    vector<Edge> edges;
    while(scanf("%d", &n) == 1 && n){
        _rep(i,1,n) nodes[i].init(i);
        root = 0;
        _for(i, 0, n-1){
            scanf("%d%d", &u, &v);
            auto &nu = nodes[u], &nv = nodes[v];
            nu.adj.push_back(v), nv.adj.push_back(u);
```

```
            nu.deg++; nv.deg++;
            if(!root && nu.deg > 1) root = u;
            if(!root && nv.deg > 1) root = v;
        }

        edges.clear();
        if(n == 2) edges.push_back(Edge{1, 2});
        else solve(edges);

        printf("%lu\n", edges.size());
        for(auto e : edges) printf("%d %d\n", e.from, e.to);
    }

    return 0;
}
```

习题 11-20 租车（Rent a Car, UVa12433）

如果你想经营一家租车公司。接下来的 N 天中已经有了一些订单，其中第 i 天需要 r_j 辆车（$0 \leqslant r_j \leqslant 100$）。初始时，你的仓库是空的，需要从 C 家汽车公司买车，其中第 i 家公司里有 c_i 辆车，单价是 p_i（$1 \leqslant c_i, p_i \leqslant 100$）。当一辆车被归还给租车公司之后，你必须把它送去保养之后才能再次租出去。一共有 R 家服务中心，其中第 i 家保养一次需要 d_i 天，每辆车的费用为 s_i（$1 \leqslant d_i, s_i \leqslant 100$）。这些服务中心都很大，可以接受任意多辆车同时保养。你的仓库很大，可以容纳任意多辆车。你的任务是用最小的费用满足所有订单。$1 \leqslant N, C, R \leqslant 50$。

例如，N=3，C=2，R=1，r={10,20,30}，c_1=40，p_1=90，c_2=15，p_2=100。d_1=1，s_1=5，最优方案是这样的：先买 50 辆车，其中在公司 1 买 40 辆，公司 2 买 10 辆，费用为 90*40+100*10=4600。第一天白天租出去 10 辆车，晚上收回之后送到服务中心保养一天，费用为 5*10=50，第 3 天白天可以再次出租。第 2 天出租 20 辆车，第 3 天把剩下的 20 辆车和保养后的 10 辆车一起出租。总费用为 4600+50=4650。

【分析】

建立源点 S 和汇点 T。对于第 i 天，建立两个结点 G_i 和 Q_i，分别表示当天的待租车库，以及待保养车库（存放当天返回的待保养车），从 G_i 到 T 建立一条边，容量为当天的市场需求 r_i，费用为 0，表示当天需要租出 r_i 辆车。从 S 到 Q_i 建一条边，容量为 r_i，费用为 0，代表从市场归还的 r_i 辆车。从 G_{i-1} 到 G_i 建立一条边，容量为 INF，费用为 0。表示汽车可以在租车公司存着不租出去。而对于每一个汽车公司，从 S 到 G_0 建立一条容量为 c，费用为 p 的边，表示第一天可以买到的车。

对于每个服务中心，因为从第 i 天市场返回来的车要在第 i+1 天送修并且在第 $i+d+1$ 天才送回租车公司供后续出租，所以从 Q_i 到 G_{i+d+1}，建立一条边（前提是 $i+d+1 < N$），容量为 INF，费用为保养费用 s。图 2.79 展示了题目描述案例对应的图的结构。



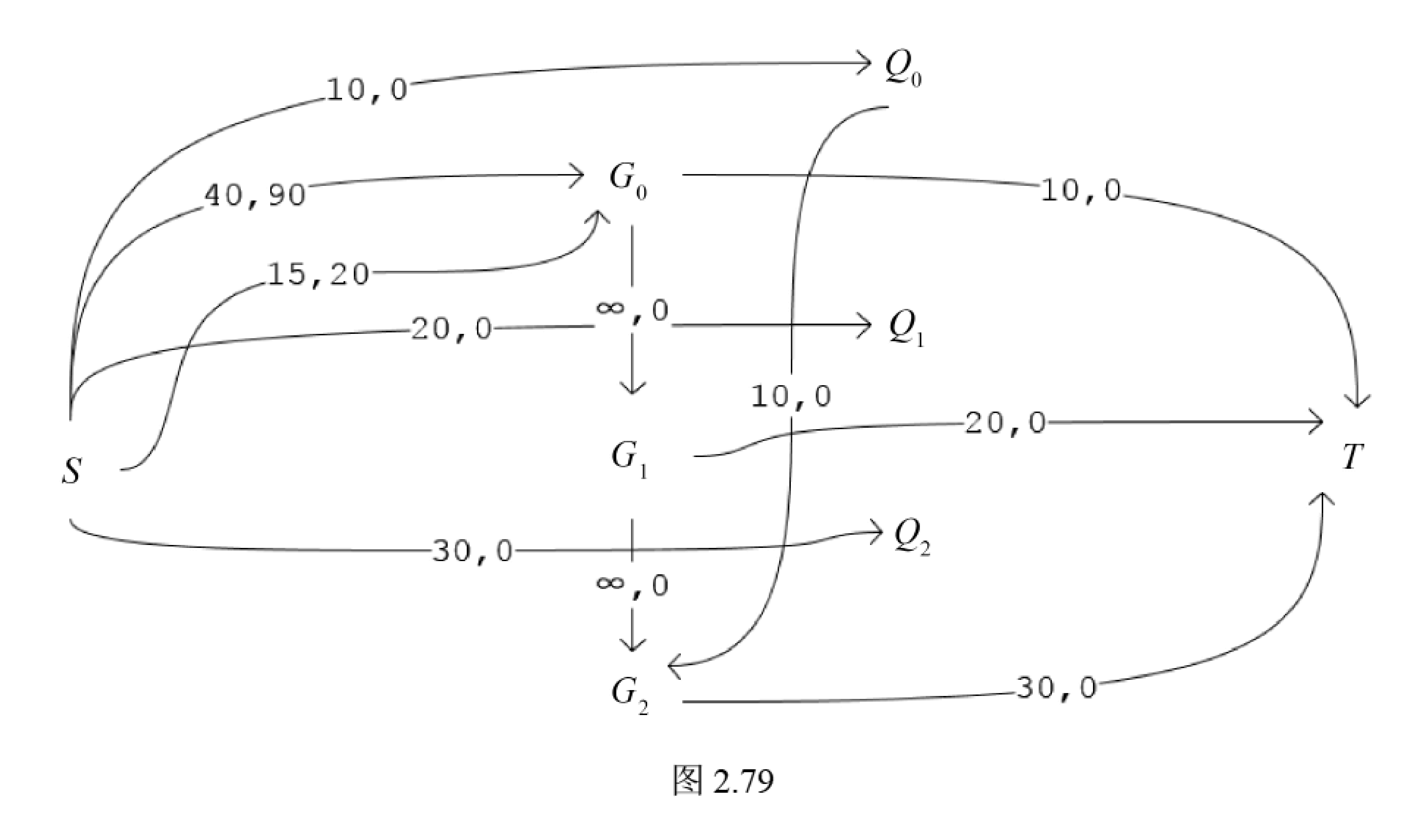


图 2.79

这样 $S \rightarrow T$ 的最小费用就是所求的最小费用，直接用 MCMF 求最小费用最大流即可，求出最小费用之后，看每条到 T 的边容量是否是最大值，也就是说满足了当天的市场需求。如果全部满足，直接输出最小费用，否则说明不满足，输出 impossible。

这个题目的难点在于：如何表示从服务中心保养返回的车，实际上这些流量是在每一天“返回”了，那么就从源点到每一天的待保养车库建一条边即可，表示流量用完之后又“回来”了。另外对于服务中心及汽车公司，虽然直观概念上是一个点，但是本质上只是有费用流量的路径而已，不用在建模时建立这个点，直接建边即可。

MCMF 算法的实现可以参考《算法竞赛入门经典——训练指南》中的 5.6.2 节。费用要使用 64 位的 long long 来存储，以免溢出。

2.10 高级专题

本节选解习题来源于《算法竞赛入门经典（第 2 版）》一书的第 12 章。

习题 12-1 自编 SketchUp（My SketchUp, Rujia Liu’s Present 4, UVa12306）

Google SketchUp 是一个很棒的软件，可以用来创建、修改和分享 3D 模型。在本题中，需要编写它的一个 2D 简化版，即 My SketchUp。

My SketchUp 的使用非常直观。例如，画两条交叉线段后，两条线段会被自动截断成 4 条，因此在图 2.80（a）中单击小圆点后只会选中一条线段（粗线部分），删除后如图 2.80（b）所示。此时单击图 2.80（b）中的小圆点，会选中另一线段。把该线段删除后剩下的两条线段会自动合并成一条线段，如图 2.80（c）所示。另外，在任何时候，重复的线段都会合并成一条。

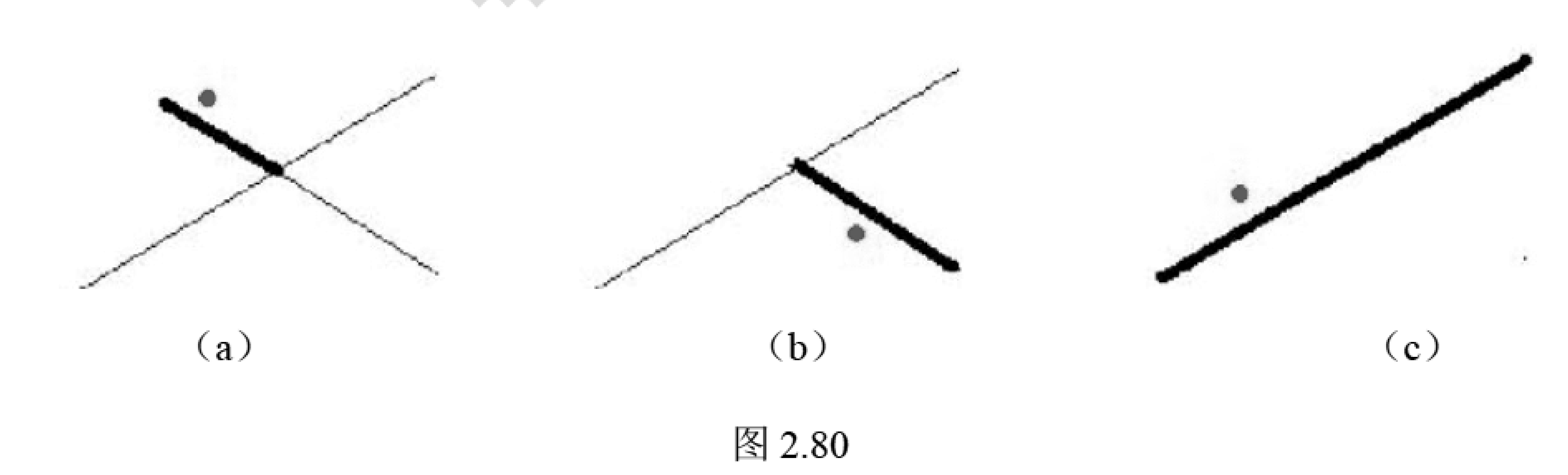


图 2.80

换句话说，对于一个图形来说，它的“长相”决定了它的实际结构，与“这个图形是如何画出来的”无关。一个图形看上去什么样的实际就是什么样的。图 2.81 所示就包含 14 个顶点和 15 条线段。

输入 n≤100 条 DRAW 和 REMOVE 语句，输出图形中的各个点的坐标和各条线段两端的点编号，按照字典序排列。

DRAW 的参数是一条折线（最多包含 20 个点），而 REMOVE 语句有 x、y 和 d 这 3 个参数，功能是删除离(x,y)距离不超过 d 的所有线段。

评注：

这是一道很考验编程能力的题目，一不注意就会让程序很复杂且容易出错。

【分析】

既然是处理二维几何点和线段，首先引入《算法竞赛入门经典——训练指南》第 4 章的 Point 数据结构以及以下几个关键函数：Dot（点积）、Cross（叉积）和 Length（向量长度）。然后建立直线和线段公用的数据结构 Line，其中包含左右端点 p1、p2。为了方便处理，两个端点要进行排序处理。保证 p1 一直在 p2 的左方或者下方。

然后回到题目本身，因为有两种操作：DRAW 和 REMOVE。前者就是在现有图形中增加一些线段，封装一个函数 addSegment，输入一个 Line 结构 L，在图中画线，实现步骤如下：

（1）检查和现有线段共线且有公共点的情况（如图 2.82 所示），将 L 和所有这样的现有线段合并形成新的 L。

（2）检测 L 和所有现有线段的交点，如果有交点，并且交点不在现有线段的两端，则现有的线段被切割成两段。删除老的线段，增加切割出来的新的线段，并且记录下来其和 L 的交点。

（3）步骤（2）中记录下来的交点将 L 切成若干段（如图 2.83 所示），作为新的线段增加进来。

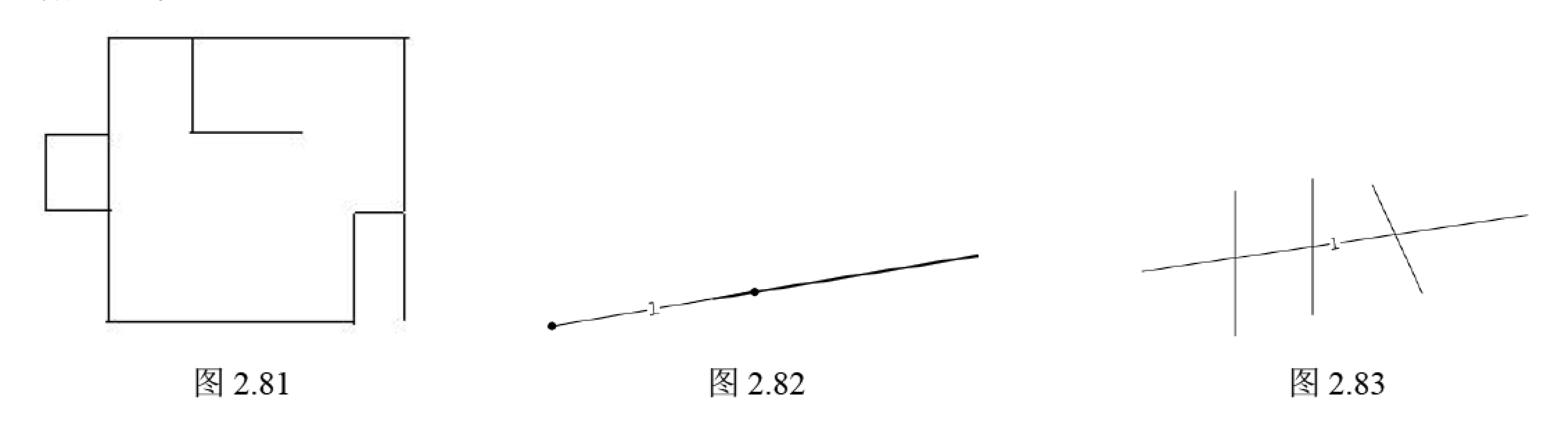


图 2.81　　图 2.82　　图 2.83

而 REMOVE 操作，实际上就是遍历所有的现有线段，把距离给定的点距离小于 d 的线段全部删除。而每次 DRAW 和 REMOVE 之后，都可能出现一些两条共线线段相连的情况，就需要将这些线段合并成一条线段。为此还需要维护每个端点以及与其相连的每条线段。

本题中因为要对所有的线段进行频繁的增删操作，使用 STL 中的链表结构也就是 list 进行线段的存储。而且 Line 的结构体较大，建议使用 list<Point>::iterator 来表示线段的引用，这样可以用来存储点对线段的引用，进行删除等操作也非常方便。参见代码中的 Picture 结构。

最后在输出结果之前，删除所有的孤立点，之后计算所有点的编号然后遍历每个点，输出以这个点为左端点的线段的左右端点编号即可。

实现上，还要封装如下函数（具体实现请参考《算法竞赛入门经典——训练指南》中的第 4 章）：

（1）判断线段是否平行或共线 isParallel。

（2）求两条直线交点 intersection。

（3）求点在直线上的位置 pointSegPos：在直线上/不在直线上/在线段上/是线段的端点。

（4）点到线段的距离 distToSeg。

主程序（C++11）如下：

```
struct Line {
   Point p1, p2;
   Line(const Point &p_1, const Point &p_2) : p1(p_1), p2(p_2) { if (p2 < p1)swap(p1, p2); };
   bool operator<(const Line&b)const { return p1 < b.p1 || (p1 == b.p1 && p2 < b.p2); }
   Vector dir()const { return p2 - p1; };
};
/*直线是否平行*/
bool isParallel(const Line &l1, const Line &l2) { return dcmp(Cross(l1.dir(), l2.dir())) == 0; }
/*直线交点*/
Point intersection(const Line &l1, const Line &l2) {
   Point v = l1.dir(), w = l2.dir(), u = l1.p1 - l2.p1;
   double t = Cross(w, u) / Cross(v, w);
   return l1.p1 + v * t;
}
enum PointLinePos {
   ON_LINE = -1, NOT_ON_LINE = 0, ON_SEGMENT = 1, ON_SEGMENT_TERMINAL = 2
};
int pointSegPos(const Line &l, const Point &p) { //点相对于线段的位置
   Vector dd = l.dir(), d1 = p - l.p1, d2 = p - l.p2;
   if (dcmp(Cross(dd, d1)) != 0) return NOT_ON_LINE;
   double a = Dot(d1, d2);
   if (dcmp(a) == 0) return ON_SEGMENT_TERMINAL;
```

```
    if (dcmp(a) < 0) return ON_SEGMENT;
    return ON_LINE;
}
double distToLine(const Line &l, const Point &p) { //点到直线的距离
    Vector v1 = l.p2 - l.p1, v2 = p - l.p1;
    return fabs(Cross(v1, v2)) / sqrt(Dot(v1, v1));
}
double distToSeg(const Line &l, const Point &p) { //点到线段的距离
    if (l.p1 == l.p2)return Length(p - l.p1);
    Vector v1 = l.dir(), v2 = p - l.p1, v3 = p - l.p2;
    if (dcmp(Dot(v1, v2)) < 0)return Length(v2);
    if (dcmp(Dot(v1, v3)) > 0)return Length(v3);
    return fabs(Cross(v1, v2)) / Length(v1);
}

typedef list<Line>::iterator ILL;
bool operator<(const ILL &a, const ILL &b) { return *a < *b; }

struct Picture {
    list<Line> lines;                       //所有的直线
    map<Point, set<ILL> > V;               //顶点，以及与这个顶点连接的线段
    map<Point, int> id;                    //顶点编号，输出用

    void newSeg(const Line &l) {           //插入新的线段
        if (l.p1 == l.p2) return;
        lines.push_front(l);
        V[l.p1].insert(lines.begin());
        V[l.p2].insert(lines.begin());
    }

    void removeSeg_IncIter(ILL &il) {      //删除线段，并且指向下一条线段
        V[il->p1].erase(il);
        V[il->p2].erase(il);
        lines.erase(il++);
    }

    void addSegment(Line l) {
        vector<Point> newPts;
        for (auto it = lines.begin(); it != lines.end();) {
            //处理 l 和现有线段共线并且有互相覆盖的情况
            if (isParallel(*it, l) && dcmp(distToLine(*it, l.p1)) == 0){
                if (it->p1 < l.p2 && l.p1 < it->p2) {
```

```
                l.p1 = min(it->p1, l.p1), l.p2 = max(it->p2, l.p2);
                removeSeg_IncIter(it);
                continue;
            }
        }
        ++it;
    }
    for (auto it = lines.begin(); it != lines.end();) {
        if (!isParallel(*it, l)) { //查看与现有所有线段的交点
            Point is = intersection(*it, l);
            int sa = pointSegPos(l, is), si = pointSegPos(*it, is);
            if (sa > 0 && si > 0) {
                if (sa == ON_SEGMENT) newPts.push_back(is); //l 被切割开
                if (si == ON_SEGMENT) {
                              //现有线段被切割，删除老的线段，增加新的线段
                    newSeg(Line(it->p1, is));
                    newSeg(Line(is, it->p2));
                    removeSeg_IncIter(it);
                    continue;
                }
            }
        }
        ++it;
    }
    newPts.push_back(l.p1), newPts.push_back(l.p2);
    sort(newPts.begin(), newPts.end());
    _for(i, 0, newPts.size() - 1) //把 l 被切割成的段挨个加进去
        newSeg(Line(newPts[i], newPts[i + 1]));
}

void removeNear(const Point& p, double d) { //删除距离 p≤d 的所有线段
    for (auto it = lines.begin(); it != lines.end();) {
        if (dcmp(distToSeg(*it, p), d) <= 0) {
            removeSeg_IncIter(it); continue;
        }
        ++it;
    }
}

void combine() { //合并所有的从一个点出发的共线线段
    for (const auto& v : V) {
        if (v.second.size() != 2) continue;
```

```
            auto i1 = *(v.second.begin()), i2 = *(v.second.rbegin());
            if (!isParallel(*i1, *i2)) continue;
            //两条线段共线
            Line l(min(i1->p1, i2->p1), max(i1->p2, i2->p2)); //两条线段合并
            removeSeg_IncIter(i1), removeSeg_IncIter(i2);
            newSeg(l);
        }
    }

    void cleanup() { //清理没有线段相连的点
        for (auto it = V.begin(); it != V.end();)
            if (it->second.empty()) V.erase(it++); else ++it;
    }

    void init() { id.clear(); lines.clear(); V.clear(); }
};

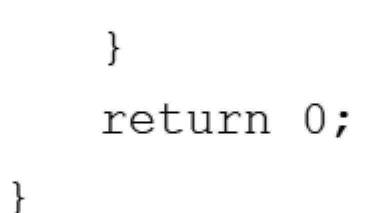
int main() {
    int n; char cmd[16]; Point p; vector<Point> ps;
    Picture pic;
    while (scanf("%d", &n) == 1 && n) {
        pic.init();
        _for(i, 0, n) {
            scanf("%s", cmd);
            if (cmd[0] == 'D') {
                ps.clear();
                while (scanf("%lf%lf", &(p.x), &(p.y)) == 2) ps.push_back(p);
                _for(j, 1, ps.size()) pic.addSegment(Line(ps[j - 1], ps[j]));
            }
            else if (cmd[0] == 'R') {
                double d;
                scanf("%lf%lf%lf", &(p.x), &(p.y), &d);
                pic.removeNear(p, d);
            };
            pic.combine();
        }
        while (strcmp("END", cmd) != 0) scanf("%s", cmd);
        pic.cleanup();
        printf("%lu\n", pic.V.size());
        int nid = 0;
        for (const auto& v : pic.V) {
            printf("%.2lf %.2lf\n", v.first.x + eps, v.first.y + eps);
```

```
            pic.id[v.first] = ++nid;
        }

        printf("%lu\n", pic.lines.size());
        for (caonst auto& v : pic.V) for (const auto& l : v.second)
            if (v.first == l->p1) printf("%d %d\n", pic.id[l->p1], pic.id
[l->p2]);
    }
    return 0;
}
int main() {
    int n; char cmd[16]; Point p; vector<Point> ps;
    Picture pic;
    while (scanf("%d", &n) == 1 && n) {
        pic.init();
        _for(i, 0, n) {
            scanf("%s", cmd);
            if (cmd[0] == 'D') {
                ps.clear();
                while (scanf("%lf%lf", &(p.x), &(p.y)) == 2) ps.push_back(p);
                _for(j, 1, ps.size()) pic.addSegment(Line(ps[j - 1], ps[j]));
            }
            else if (cmd[0] == 'R') {
                double d;
                scanf("%lf%lf%lf", &(p.x), &(p.y), &d);
                pic.removeNear(p, d);
            };
            pic.combine();
        }
        while (strcmp("END", cmd) != 0) scanf("%s", cmd);
        pic.cleanup();
        printf("%lu\n", pic.V.size());
        int nid = 0;
        for (const auto& v : pic.V) {
            printf("%.2lf %.2lf\n", v.first.x + eps, v.first.y + eps);
            pic.id[v.first] = ++nid;
        }

        printf("%lu\n", pic.lines.size());
        for (const auto& v : pic.V) for (const auto& l : v.second)
            if (v.first == l->p1) printf("%d %d\n", pic.id[l->p1], pic.id
[l->p2]);
```



```
    }
    return 0;
}
```

习题 12-18　谱曲（Melod[y] "Creation", Rujia Liu’s Present 6, UVa12566）

可以用字符串来表示一个简谱，其中小节线为“|”，s1=s2 表示一个转调，即该音符在转调前是 s1，转调后是 s2。例如，下面的简谱是一个“诡异版”的生日歌：

5 5 6 5 1=4 3 | 1 1 2 1 5 4 | 1=5 5 5 3 1 7=3 2 | b7 b7 6 4 5=2 1 | |

输入一个简谱，要求将它改写，使得升降号不超过 k 个，在此前提下转调的次数最少。多解时输出字典序最小的。要求音符数不超过 100。

相关的音乐背景知识：

首调唱名法中有 12 个不同的音节：1, #1/b2, 2, #2/b3, 3, 4, #4/b5, 5, #5/b6, 6, #6/b7, 7。两个相邻音节之间的音程是一个半音。其中，#1 表示 1 升 1 个半音，b2 表示 2 降 1 个半音。

你可以标记这 12 个音符为 0～11，任意两个音符间音程的计算是通过相减后取模 12 来进行。例如，#6 和 2 之间的音程就是 2−10=4(mod 12)。也可以数出来：#6→7→1→#1→2，刚好 4 个半音。

如果听到演奏“1 2 3”，同样可以认为是“4 5 6”，因为两段旋律的相邻音程都是“2 2”。同样，“2 3 4 5”和“6 7 1 2”（后面的“1 2”是高一个八度的）也是相似的，而且两段旋律的相邻音程都是(2 1 2)。这里“相似”指的是可以把一段旋律改成另一段。

考虑最后一个输入案例：“1 #1 2 #2 3 #4”，其音程序列是（1, 1, 1, 1, 2）。重写的谱子可以包含一次转调，所以可以分成两部分：“2 #2 3 4=3”和“4=3 4 5”，第一部分的音程序列是（1,1,1），第二部分是（1,2）。注意，第一部分中的“4=3”是指转调前的“4”，第二部分“4=3”指的是转调后的“3”。另外一种同样使用了最小的转调次数，但是按照字典序更大的是“3 4 #4 5=3 4 5 ||”。

【分析】

本题音乐背景知识较多，题意的关键点是：对于输入乐谱，计算出最优转调次数最少，且升降号不超过 k 个的写法。前提是这个写法中相邻音符之间以半音个数计算的音阶，必须和输入乐谱相同。

定义状态 S(i,j,k)，其中：

（1）i 表示已经确定 i～N−1 的每个音符的写法，包含是否变调的信息，[0, i−1]还未确定。

（2）j 表示确定过的音符中出现了小于等于 j 次的变音记号（“#/b”）。

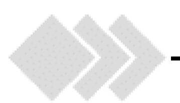
（3）k 表示第 i 个音符在变调前相对于 1 的音阶是 k 个半音。

定义如下所示。

（1）dp[s]：已经确定的音符中变调的次数，也就是“=”的次数，则有 dp[N]=0。

（2）opt[s]：dp[s]是最优解时，第 i 位的音符字符串表示。

（3）pj[j], pk[k]：dp[s]最优时，对应的前一个 i+1 的最优状态是(i+1, pj[j],pk[k])，输出时使用。

为了保证字典序，选择从后向前递推，也就是从前到后决策。就是从后到前遍历第 *i* 位所用的音符以及是否变调，变成什么调的各种情况，同时更新上文定义的各个数组即可。算法的时间复杂度为 O(NM)。完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 104, INF = 0x3f3f3f3f, NS = 16;
vector<string> SYL[NS]; //音高到音符的映射
unordered_map<string, int> STEP = { //音符相对于 1 的音高
   {"1",  0}, {"#1", 1}, {"b2", 1}, {"2", 2}, {"#2", 3},
   {"b3", 3}, {"3", 4}, {"4", 5}, {"#4", 6}, {"b5", 6}, {"5", 7}, {"#5", 8},
   {"b6", 8}, {"6", 9}, {"#6", 10}, {"b7", 10}, {"7", 11}
};
void initSyls() { for (const auto& p : STEP) SYL[p.second]. push_back
(p.first); }
struct ACC { //输入乐谱上面每个位置的音符
   bool afterBar; //是否在'|'之后
   int stepBefore, stepAfter;          //转调前后的音高（如果没有转调，二者相同）
   ACC() : afterBar(false), stepBefore(0), stepAfter(0) {}
};

int N, M;
vector<ACC> trans;                          //输入的乐谱
int dp[MAXN][2 * MAXN][NS], pj[MAXN][2 * MAXN][NS], pk[MAXN][2 * MAXN][NS];
string opt[MAXN][2 * MAXN][NS];

void solve() {
   memset(dp, INF, sizeof(dp));
   memset(dp[N], 0, sizeof(dp[N]));    //还没确定任何音符，自然是 0

   for (int i = N - 1; i >= 0; i--) _rep(j, 0, M) _for(k, 0, 12) {
      //依次考虑每个位置，j : 变调的次数，k : 12 个音阶
      int delta = (trans[i + 1].stepBefore - trans[i].stepAfter + 12) % 12;
//音符之间的音阶
      if (j > 0 && dp[i][j - 1][k] != INF) {
         dp[i][j][k] = dp[i][j - 1][k];
         opt[i][j][k] = opt[i][j - 1][k];
         pj[i][j]a[k] = pj[i][j - 1][k], pk[i][j][k] = pk[i][j - 1][k];
      }

      for (const auto& sl : SYL[k]) { //尝试 step = k 各种音符，进行变调
         string s = sl;
         int d = (sl.length() == 2), ni = i+1, nj = j-d, nk = (k+delta) % 12;
```

```
            auto updaateD = [&i, &j, &k, &s, &nj, &nk](int nd) {
             //更新 DP 状态函数
                int &d = dp[i][j][k];
                if (nd < d || (nd == d && s <= opt[i][j][k]))
                    d = nd, opt[i][j][k] = s, pj[i][j][k] = nj, pk[i][j][k] = nk;
            };

            //不变调，状态转移
            if (nj >= 0 && dp[ni][nj][nk] != INF) updateD(dp[ni][nj][nk]);

            //变调的状态转移
            _for(mk, 0, 12) for (const auto& msl : SYL[mk]) {
                //mk:变调之后的音阶，msl：变调后的音符
                s = sl + "=" + msl;
                ni = i + 1, nj = j - (d + (msl.size() == 2)), nk = (mk + delta) % 12;
                if (nj >= 0 && dp[ni][nj][nk] != INF) updateD(dp[ni][nj][nk] + 1);
            }
        }
    }

    int ans = INF, k = 0;
    _for(i, 0, 12) {
        int d = dp[0][M][i];
        if (d == INF) continue;
        if (d < ans || (d == ans && opt[0][M][i] < opt[0][M][k])) ans = d, k = i;
    }

    function<void(int, int, int)> output = [&](int i, int j, int k) {
        if (i == N) return;
        cout << opt[i][j][k] << " ";
        if (trans[i].afterBar) cout << "| ";
        output(i + 1, pj[i][j][k], pk[i][j][k]);
    };
    output(0, M, k);
    cout << "||" << endl;
}

int main() {
    initSyls();
    int T; string s; ACC acc;
    scanf("%d", &T);
    _rep(t, 1, T) {
```

```
        cout << "Case " << t << ": "; cin >> M; trans.clear();
        while (cin >> s && s != "||") {
            if (s == "|") { trans.back().afterBar = true; continue; }
            size_t ep = s.find('=');
            if (ep != string::npos)
                acc.stepBefore = STEP[s.substr(0, ep)],
                    acc.stepAfter = STEP[s.substr(ep + 1)];
            else
                acc.stepAfter = acc.stepBefore = STEP[s];
            trans.push_back(acc);
        }
        N = trans.size();
        trans.push_back(ACC());
        if (M > 2 * N) M = 2 * N; //不可能超过 2*N 个符号
        solve();
    }
    return 0;
}
```

第 3 章　比赛真题分类选解

3.1　搜　　索

泡泡龙（Puzzle & Dragons, ACM/ICPC Asia-Xian 2014, LA7036，难度 6）

在泡泡龙游戏中，有一个 5*6 的方阵，每个格子包含一个珠子。珠子有 6 种类型：F、W、P、L、D 和 C。

游戏开始时，玩家可以选择一个珠子并且沿着一个路径移动。路径上的第一个珠子会移动到第二个的位置上，第二个会移到第三个，以此类推。最后一个珠子会移动到第一个的位置上。珠子只能沿上下左右 4 个方向移动。例如，如果某一行原来是“FWPLDC”，第一个珠子拿起来一直移到最右边之后，这一行就变成“WPLDCF”。

在选定路径并移动珠子之后，就开始消除。如果同一行/列有 3 个或以上的同样的珠子连起来（称为链），就会消除。并且在这个链上方的珠子就会落下来，消除之后不会再出现新的珠子。

如果在移动之后有多个链可被消除，它们就形成“组合（Combo）”。注意，两个同样珠子组成的链相邻或者连起来，就会被统计成一个组合，图 3.1 就是两个例子。

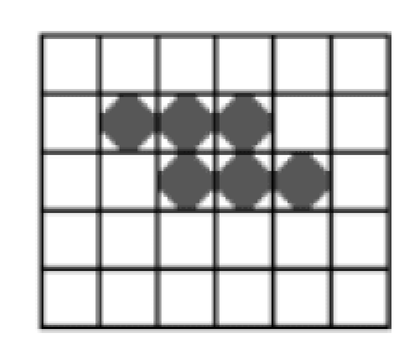
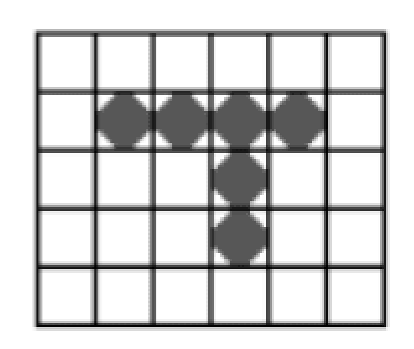



图 3.1

给出方阵上每个格子放的珠子类型，需要选择一条长度不长于 9 的路径，使得组合的数量最多。如果有多个，就选择能够消除最多珠子的路径。如果依然有多个，选择任意一个路径长度最短的即可。输出这个路径能够消除的组合数量以及路径的长度，最后输出路径上第一个珠子的坐标（方阵左上角坐标是(1,1)，右下角是(5,6)），以及这个珠子每一步移动的方向（用‘UDLR’4 个字符之一表示，分别是上下左右）。

【分析】

本题是一个已经限制了深度的搜索问题（≤9），首先是遍历所有的路径起点，起点确定后依次对每一步走的方向进行回溯，注意第二步开始就不能再回头走了。每走一步就对形成的路径尝试进行消除，然后用消除后的结果更新最优答案。注意步数不能超过 9。

消除过程如下：

（1）Grid 复制一份副本。

（2）在副本中查找并标记所有连续 3 个在同一行/列的同色珠子，这些珠子都是待消除



区域。

（3）标记完成之后就使用 DFS 消除每个区域，并对所有的 combo 区域进行计数。

（4）将珠子消除后形成的空格上方形成的珠子往下平移。

（5）只要还有 combo 存在，就再次消除，直到所有的 combo 消除完毕为止。

完整程序（C++11）如下：

```
using namespace std;

struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
  Point& operator=(const Point& p){ x = p.x; y = p.y; return *this; }
};
typedef Point Vector;
Vector operator+ (const Vector& A, const Vector& B) { return Vector(A.x+B.x, 
A.y+B.y); }

struct Path {
    int combo, drop, length;
    Point start;
    char solution[16];
    void init(int len = 0, const char* str = nullptr){
        start.x = 0, start.y = 0, combo = 0, drop = 0, length = len;
        if(str) memcpy(solution, str, len);
        solution[len] = 0;
    }

    bool operator<(const Path& p) const {
        if(combo != p.combo) return combo < p.combo;
        if(drop != p.drop) return drop < p.drop;
        return length > p.length;
    }
};

const int N = 5, M = 6;
const vector<Vector> DIRS = {{0,1}, {0,-1}, {1,0}, {-1,0}};
const char op[5] = "RLDU";
char Buf[N][M+1], Grid[N][M+1], sol[11];
bool ELIM[N][M];
Path ans;
bool valid(const Point& p){ return p.x >= 0 && p.x < N && p.y >= 0 && p.y 
< M; }
```

```
//dfs 消除点 p 周围所有颜色为 c 的格子，返回被消除的数量
int eliminate(const Point& p, char c){
    if(!valid(p)) return 0;
    if(Buf[p.x][p.y] != c || !ELIM[p.x][p.y]) return 0;
    Buf[p.x][p.y] = ' ', ELIM[p.x][p.y] = false;
    int res = 1;
    for(const auto& d : DIRS) res += eliminate(p + d, c);
    return res;
}
bool eliminate(int& combo, int& drop){ //消除所有能消除的
    bool any = false;
    combo = drop = 0;
    _for(i, 0, N) _rep(j, 0, M-3){      //3 个一行
        char c = Buf[i][j];
        if(c == ' ') continue;
        if(c == Buf[i][j+1] && c == Buf[i][j+2])
            any = ELIM[i][j] = ELIM[i][j+1] = ELIM[i][j+2] = true;
    }

    _rep(i, 0, N-3) _for(j, 0, M){      //3 个一列
        if(Buf[i][j] == ' ') continue;
        if(Buf[i][j] == Buf[i+1][j] && Buf[i][j] == Buf[i+2][j])
            any = ELIM[i][j] = ELIM[i+1][j] = ELIM[i+2][j] = true;
    }
    if(!any) return false;
    _for(i, 0, N) _for(j, 0, M){
        if(!ELIM[i][j]) continue;
        combo++;
        drop += eliminate(Point(i,j), Buf[i][j]);
    }

    _for(j, 0, M){                    //上面的珠子往下平移
        int bottom = N-1;
        for(int i = N-1; i >= 0; i--){
            if(Buf[i][j] == ' ') continue;
            if(bottom != i){
                Buf[bottom][j] = Buf[i][j];
                Buf[i][j] = ' ';
            }
            bottom--;
        }
```



```
    }
    return true;
}
//看看路径的消除结果
void tryPath(int step, const Point& st){
    Path res;
    res.init(step, sol);
    res.start = st;
    memcpy(Buf, Grid, sizeof(Grid));
    int combo, drop;
    while(eliminate(combo, drop)) res.combo += combo, res.drop += drop;
    if(ans.length == 0 || ans < res) ans = res;
}
//寻找路径:(路径上下一个点，已经走出的步数，上一个方向，路径的起点)
void findPath(const Point& p, int step, int lastDir, const Point& st){
    tryPath(step, st);
    if(step >= 9) return;
    _for(i, 0, 4){
        if(step >= 2 && lastDir == (i^1)) continue; //第 2 步就不能再回头走
        Point np = p + DIRS[i];
        if(!valid(np)) continue;
        swap(Grid[p.x][p.y], Grid[np.x][np.y]);
        sol[step] = op[i];
        findPath(np, step+1, i, st);
        swap(Grid[p.x][p.y], Grid[np.x][np.y]);
    }
}

int main(){
    int T; scanf("%d", &T);
    _for(t, 0, T){
        _for(i, 0, N) scanf("%s", Grid[i]);
        ans.init();
        //遍历路径的起点
        _for(x,0,N) _for(y,0,M) findPath(Point(x,y), 0, 4, Point(x,y));
        printf("Case #%d:\n", t+1);
        printf("Combo:%d Length:%d\n%d %d\n%s\n",
            ans.combo, ans.length, ans.start.x+1, ans.start.y+1, ans.solution);
    }

    return 0;
}
```

另一个 n 皇后问题（Another n-Queen Problem, UVa11195，难度 7）

相信 n 皇后问题对每个研究回溯法的人来讲都不陌生，这个问题是要在一个 $n*n$ 大小的棋盘上摆 n 个皇后，让她们不会互相攻击。为了让这个问题更难一点，设计了一些障碍物在棋盘上，在这些点上不能放皇后，这些障碍物并不能防止皇后被攻击。

在传统的八皇后问题中，旋转与镜射被视为不同解法，因此有 92 种可能的方式来放置皇后。输入棋盘的大小 n（$3 \leqslant n \leqslant 15$），以及 n 行表示棋盘布局的长度为 n 的字符串，其中空格用“.”表示，障碍物占用的格子用“*”表示。计算并输出 n 皇后问题的解。

【分析】

本题的搜索逻辑和经典的八皇后（参考《算法竞赛入门经典（第 2 版）》中的 7.4.1 节）问题基本相同。不同点在于：n 比较大，经典解法来解决会直接超时。注意到 $n \leqslant 15$，可以使用一个整形作为位向量来分别表示列以及两个对角线上的每个位置是否可以放皇后。每一层使用位运算来求出这一行所有可放皇后的位向量 can，然后使用一个位运算技巧：使用 x&(−x)可以快速得出一个数，这个数字只在对应 x 的最右边的 1 的位置包含一个 1。然后就可以尝试在这个位置放置皇后，继续下一层搜索。

完整程序如下：

```
using namespace std;
const int MAXN = 32;
int N, rows[MAXN], ans;
/*
    r 是当前的行号
    v 是一个位集合，其第 b 位为 1 表示棋盘的第 b 列还没有放皇后，可以继续再放
    d1 是表示正对角线（左上右下），其中第 b 位表示 r+c=b 的那条对角线是否可用，可用为 1
    d2 是表示斜对角线（右下左上）的位集合，其中第 b 位表示 r-c+N-1=b 的那条对角线是否可
以放置，可用为 1
*/
void dfs(int r, int v, int d1, int d2) {
    if(r == N) { ans++; return; }
    int can = rows[r] & v & (d1>>r) & (d2>>(N-r-1));
    /*
        当确定了 r 之后
        d1>>r 的第 i 位表示第 r 行第 i 列所在的正对角线上是否可以放皇后
        d2>>N-r-1 的第 1 位表示，第 r 行第 i 列所在的反对角线上是否可以放皇后
        这样直接用位运算就快速求出了所有可以放皇后的列的集合
    */
    while(can) {
        /* 这里使用了一个位运算技巧：使用 x & (-x)可以求出一个数，这个数字的二进制只包
含一个 1，这个 1 的位置对应于 x 中最右边的 1 */
        int x = can & (-can);
        dfs(r+1, v^x, d1^(x<<r), d2^(x<<(N-r-1)));
        can ^= x;
    }
```

```
}

int main() {
    char buf[MAXN];
    for(int t = 1; scanf("%d", &N) == 1 && N; t++){
        _for(i, 0, N){
            rows[i] = (1<<N)-1;
            scanf("%s",buf);
            _for(j, 0, N) if(buf[j]=='*') rows[i] ^= (1<<j);
        }
        ans = 0;
        dfs(0,(1<<N)-1,(1<<(2*N-1))-1,(1<<(2*N-1))-1);
        printf("Case %d: %d\n", t, ans);
    }
    return 0;
}
```

生日蛋糕（Birthday Cake, UVa11196，难度 8）

需要做一个蛋糕，包含正好 m（m<11）层同心圆柱体，从下到上的每一层编号是 1,2,⋯ m。第 i 个圆柱的半径是正整数 r_i，高度是正整数 h_i，而且对于所有的 i<m 有：$r_i > r_{i+1}$ 且 $h_i > h_{i+1}$。要求总的体积为 $n\pi$（n<100001）。除了底面之外的所有的表面都要涂上冰淇淋，需要涂冰淇淋的尽量少，也就是除了底面之外的表面积尽量小。计算每一层的高度和半径使得表面积最小。输出 S，$S\pi$ 为表面积最小值。问题无解则输出 0。

【分析】

不难想到，从下到上对每一层的半径和高度进行回溯，看看最终能不能做出符合要求的蛋糕。下文为描述方便，记从上到下的每一层编号依次为 1～m。则总体积 V 和涂冰淇淋的表面积分别为：$V=\sum_{i=1}^{m} r_i^2 h_i$，$S=r_m^2+\sum_{i=1}^{m} 2r_i h_i$ 。

题中已经说明了 $1 \leqslant h_k, r_k \geqslant k$。但是如果只有这一个限制条件，时间复杂度就是 $m!$。必然会超时，考虑剪枝优化。当搜索到第 k 层时，记当前的状态为：V（剩下 1～k 层可以用的蛋糕体积），S（M～k-1 已经使用的表面积），H（第 k 层的高度上限），R（第 k 层的半径上限）。则考虑如下剪枝优化：

（1）剩下 k 层所要用的体积最少为：$\sum_{i=1}^{k} r_i^2 h_i \geqslant \sum_{i=1}^{k} i^3 = k^2(k+1)^2/4$。故只有当剩余的体积 $V>k^2(k+1)^2/4$ 时才继续搜索。

（2）同理剩下 k 层所要使用的表面积为：$S_k = 2\sum_{i=1}^{k} r_i h_i \geqslant \sum_{i=1}^{k} i^2 = \dfrac{k(k+1)(2k+1)}{3}$。故只有当 $S+\dfrac{k(k+1)(2k+1)}{3} < \text{best}$ 才继续往下搜索，其中 best 是当前已经计算出的最优解。

（3）$V_k=\sum_{i=1}^{k} r_i^2 h_i < r_k \sum_{i=1}^{k} r_i h_i = r_k s_k/2$，所以有 $S_k > \dfrac{2V}{r_k}$。所以只有当 $\dfrac{2V}{r_k}+S<\text{best}$，才有可能搜出一个更优解。

（4）确定 r_k 之后，在遍历 h_k 的可能值时，上界自然是 min(V/(r*r), H)，而下界要满足：

$$V=\sum_{i=1}^{k} r_i^2 h_i < r_k^2 \sum_{i=1}^{k} h_i \leqslant r_k^2 \sum_{i=1}^{k}(h_k-k+i)=r_k^2(2h_k-k+1)k/2$$

也就是说 $V < r_k^2(2h_k-k+1)k/2$。

☞注意：

推导过程中用到了一些平方和立方数列的求和公式，如下所示。

$$\sum_{i=1}^{n} i=\frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3=\left[n(n+1)/2\right]^2$$

星际争霸中的挖矿（Mining in Starcraft, UVa12742，难度 10）

在星际争霸游戏中，有两种资源：矿石和气。你可以给农民（SCV）发指令来挖矿或气：

（1）如果发一个挖矿指令给 SCV，可以在 $t1$ 单位时间后得到 8 个单位的矿石。

（2）如果是挖气指令，可以在 $t2$ 后得到 8 个单位的气。

只能在农民完成上个指令之后再发新的指令。

可以使用 50 个单位的矿费时 $t3$ 建造一个农民。一旦开始建造就要花去 50 矿石。并且同一时间只能造 1 个农民，也就是说上一个没造完就不能造下一个。

一开始你有 50 个单位的矿石和 4 个农民，计算要挖到 $p1$ 的矿和 $p2$ 的气需要的最短时间 T。同时要输出一个挖矿的计划（如果有多个，任选一个输出）。计划的每一行按照如下格式输出。

- *t* 0：在时间 *t* 建造一个新农民。
- *t i* 1：在时间点 *t* 给农民 *i* 发一个挖矿指令。
- *t i* 2：在时间点 *t* 给农民 *i* 发一个挖气指令。

农民的编号是 1,2,3…。一开始的农民编号为 1…4，新造的农民编号按照建造顺序依次递增。

在 T 时刻，所有的农民必须已经闲下来，并且没有正在建造的农民。输入数据范围：$1 \leqslant t1,t2,t3 \leqslant 10$，$0 \leqslant p1,p2 \leqslant 100$。

【分析】

题目给人的第一感觉是相关的变量太多，如果都考虑进去，搜索空间巨大，无法在规定时间内完成。

可以对所用的时间上限进行二分查找。初始时间的上限很容易根据已有的 SCV 个数以及 $p1$、$p2$ 计算出来。每次查找时，首先固定 SCV 的个数（包括一开始的 4 个）。搜索时依次决策每个 SCV 需要收集的 GAS 数量，决策完成之后，就可以根据固定下来的条件对每个 SCV 建造的时间以及每个时间点的挖矿数量进行模拟，同时记录相关的命令。最后排序输出即可。



注意以下几点：

（1）只有在二分查找到最后一层才需要在模拟时记录所有的命令，可以节省大量时间。

（2）任何一个可行解，都可以把挖矿改到前面，仍然可行，反过来就不一定了，因为可能需要用挖来的矿造农民。

（3）新造的农民，每个都至少要挖矿一次，因为如果这个农民不采矿把自身费用赚回来，那么一开始的矿石就不够了。

完整程序如下：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)

struct CMD {
    enum Type { Build = 0, Minearl = 1, Gas = 2 };
    int time, cmd, id;
    CMD(int t = 0, int c = Build, int i = 0) : time(t), cmd(c), id(i) {}
    bool operator<(const CMD& c) const {
        if (time != c.time) return time < c.time;
        if (cmd != c.cmd) return cmd < c.cmd;
        return id < c.id;
    }
};
ostream& operator<<(ostream& os, const CMD& c) {
    os << c.time;
    if (c.cmd != CMD::Build) os << ' ' << c.id;
    return os << ' ' << c.cmd;
}
const int MAXT = 50, MAXK = 20;
//最长的挖矿时间，最大农民个数，因为需要挖气的次数 G+1<14，所以写 20 了
int t1, t2, t3, p1, p2;
//挖矿单位时间，挖气单位时间，造 1 个农民所需时间，目标的矿数，目标的气数
bool found = false;

int mineCnt[MAXT+4], gasTime[MAXK];
vector<int> newSCV[MAXT+4];
/*
    实际上是复杂的 IDFS
    i : 对第 i 个农民开始挖气的时间进行决策
    G : 总共还要挖多少次气，对于 i+1 ~ SCV 的农民来说
    genCmds : 用来节省时间，是否生成命令序列
    M : 时间上限
    SCV : 要达到的农民的总的个数
    cmds : 所有的命令序列
*/
```

```
    void dfs(int i, int G, bool genCmds, int TEnd, const int SCV, vector<CMD>&
cmds) {
        //assert(i <= SCV);
        if (i < SCV) { //g: 遍历，编号为 i 的农民要负责采几次气
            for(int g = (i == SCV-1 ? G : (i>=4));  g <= G && TEnd-t2*g >= 0; g++){
                gasTime[i] = TEnd - t2*g;   //农民 i 开始挖气的最晚时间
                dfs(i+1, G-g, genCmds, TEnd, SCV, cmds);
                if(found) return;
            }
            return;
        }

        if (genCmds) cmds.clear();
        fill_n(mineCnt, MAXT, 0);
        _for(t, 0, MAXT + 4) newSCV[t].clear();
        int sCnt = 4;                       //SCV 个数
        _for(i, 0, sCnt) newSCV[0].push_back(i);
        mineCnt[0] = 50;                    //一开始有 50 个单位的矿
        bool valid = true;
        int mineSum = 0, lastScvT = 0;      //总挖矿数；最后一个造出的
        _for(t, 0, TEnd+1) {                //对每一秒钟的情况进行模拟
    //造不到提前预设好的农民个数了
            if (sCnt < SCV && t + t3 > TEnd) { valid = false; break; }
            for (const auto id : newSCV[t]) { //第 t 秒新造出来的农民 id
                for (int tid = t; tid + t1 <= gasTime[id]; tid += t1) {
    //id 在 t 到 gast[id]秒之前全部挖矿
                    mineCnt[tid+t1] += 8;   //挖到 8 个单位的矿
                    if (genCmds) cmds.push_back(CMD(tid, CMD::Mineral, id + 1));
                }
            }
            mineSum += mineCnt[t];
            if (sCnt < SCV && mineSum >= 50 && lastScvT <= t) {
    //农民还没造够，并且有矿，且最后一个农民已经造好
                mineSum -= 50;
                if (t + t3 > gasTime[sCnt]) { valid = false; break; }
    //造出来的农民来不及挖气了
                lastScvT = t + t3;
                newSCV[t + t3].push_back(sCnt++); //一有钱就造农民，贪心最优策略
                if (genCmds) cmds.push_back(CMD(t, CMD::Build, sCnt));
            }
        }
    //农民数量一定要造够，主程序里面枚举了 SCV 个数，避免同一个方案考虑多次
        if (valid && mineSum >= p1 && sCnt == SCV) {
            found = true;
            if (genCmds) {
```



```
            _for(s, 0, SCV)
                for (int t = gasTime[s]; t < TEnd; t += t2)
                    cmds.push_back(CMD(t, CMD::Gas, s + 1));
        }
    }
}

int main() {
    vector<CMD> cmds;
    for (int T = 1;  cin >> t1 >> t2 >> t3 >> p1 >> p2 && t1 > 0; T++) {
        int L = -1, R = MAXT, G = (p2+7) / 8; //向上取整，还需要挖多少次气
        bool last = false;
        while (true) {
            int M = last ? R : (L + R) / 2; //时间上限是 M
            for (int SCV = 4; SCV <= max(G + 1, 4); SCV++) {
//总共有多少个农民
                found = false;
                memset(gasTime, -1, sizeof(gasTime));
                dfs(0, G, last, M, SCV, cmds);
                if (found) break;
            }
            if (last) break;
            if (found) R = M; else L = M;
            if (L + 1 == R) last = true;
        }
        cout<<"Case "<<T<<": "<<R<<endl;
        sort(cmds.begin(), cmds.end());
        for(const auto& c : cmds) cout<<c<<endl;
        cout<<endl;
    }
    return 0;
}
```

3.2 模　　拟

相交的日期区间（Intersecting Dates, World Finals 2004 – Prague, LA2997，难度 2）

开发一个程序来从一个服务商那里获取股票历史数据，有些特定日期的股票数据有重复获取的情况，所以需要维护所有已经查询过的历史股票数据。当有新的查询请求时，就可以尽量从保存过的已经查询过的数据中获得以减少成本：

（1）给出所有已经进行过的查询，格式如下：d1 d2。d1 和 d2 的格式都是 YYYYMMDD。

其中，YYYY 表示年，范围是 1700～2100，MM 和 DD 分别表示月和日。d1 和 d2 表示之间的每一天都已经查询过并且保存了结果。

（2）给出所有的新的查询，格式同上，为 d1,d2，表示需要查询之间每一天的股票数据。

现在需要计算出，到底有哪些日期的数据需要重新从服务器获得。按照递增的顺序输出每一个这样的时间段。

【分析】

首先要在已知日期区间输入之后保存下来。然后有新查询请求时，过滤掉已经保存的日期，并且把所有需要查询的日期切分成多个连续区间输出。实现上可以使用从 17000101 开始的天数作为代表来存储每个日期，这样就可以把日期对应成一个整数。初始还要建立一个整数到日期，以及日期到整数的双向对应关系。

大调问题（A Major Problem, World Finals 2001 - Vancouver, LA2237，难度 3）

西洋音乐中有 12 个常用音符，使用 A～G 的大写字母来表示，字母后附上一个升调符号“#”或者降调符号“b”。如下所示，斜杠表示一个音符的两种写法：

C/B#　C#/Db　D　D#/Eb　E/Fb　F/E#　F#/Gb　G　G#/Ab　A　A#/Bb　B/Cb

任意两个相邻音符之间的距离称为半个音程。两个音符之间间隔 1 个的距离为全音程。一个大调音阶由 8 个音符构成。从上述音符中挑选任何一个作为开始，从左到右选择音符，音程依次是“全 全 半 全 全 全 半”，（全代表一个全音，半代表一个半音），如果选到最右边就从左边第一个音符继续开始。例如，以 C 和 Db 为起点的大调音阶就是：

C　D　E　F　G　A　B　C

Db　Eb　F　Gb　Ab　Bb　C　Db

大调音阶的选取要遵循以下规则：

（1）音阶中，每一个 A～G 之间的字母必须出现并且只能出现一次，但是首尾字母可以是相同的。

（2）音阶中不能同时出现#和 b。

音阶的第一个音符称为音阶的主音，上述音阶的主音就是 C 和 Db。把一个音符从一个音阶转换到另外一个音阶，就是要找到这个音符在第一个音阶中的位置，然后寻找在第二个音阶中对应位置的音符即可。

输入两个音阶的主音，以及一系列的音符，把这些音符从第一个主音转换到第二个主音。

需要注意的是，按照大调音阶的选取规则，某些音符无法作为主音选取出 8 个音符组成一个音阶。

【分析】

尝试每一个音符作为主音，对后续使用的每个音符进行回溯，每一步考虑使用过的字母以及是否已经包含#和 b，这样可以预处理出所有的合法主音以及对应的大调音阶，然后对每一个输入首先判断输入的主音是否合法。如果都合法，则对于输入的每个音符，首先



查找其在第一个主音对应的大调音阶中的位置，然后查找第二个主音的大调音阶中对应位置上的音符即可。所有的合法主音以及对应的大调音阶使用 map 来存放。

欧元的扩散（Eurodiffusion, World Finals 2003 - Beverly Hills, LA2724，难度 3）

欧元区国家使用的纸币都必须是完全相同的，但每个国家都可以制造印有自己国家图案的硬币，而且这些个性化的硬币也可以在欧元区的所有国家流通。

一种类型的硬币由一个国家铸造，然后逐步扩散到其他国家。现在需要写程序来模拟特定面值的硬币在整个欧元区的扩散过程。这里使用一种简化的模型：只考虑一种面额的硬币。给出一个矩形的平面网格，所有的城市都在格点上，每个城市最多有上下左右 4 个相邻城市。每个城市只属于一个国家，而且一个国家的城市在平面上刚好组成一个矩形。如图 3.2 所示是一个 3 个国家，28 个城市的地图。

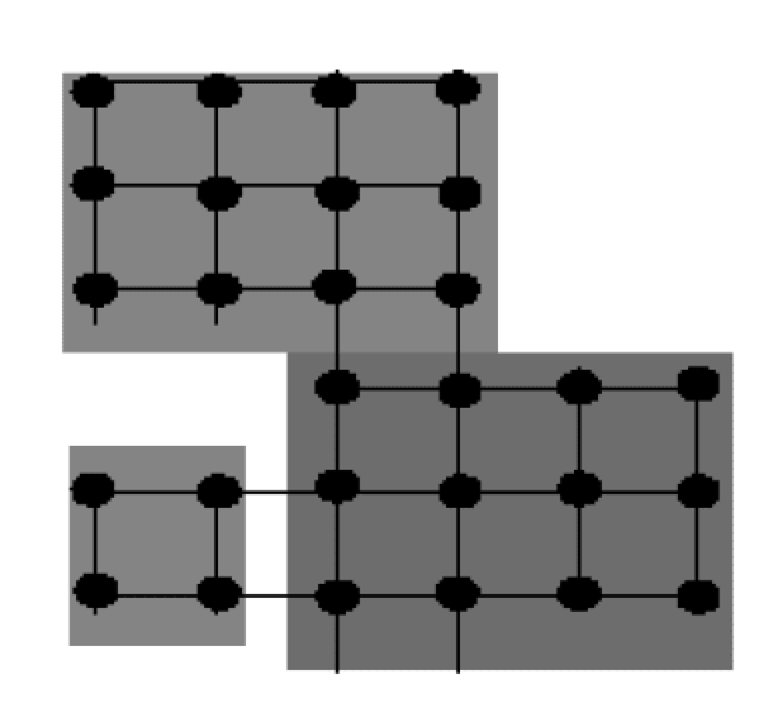


图 3.2

地图上每个国家都是连通的，但是国家之间可能有洞，表示海洋或者非欧元区。一开始每个城市都只有 1000000 个本国硬币。每天一开始按照上一天的余额，对于拥有的每个国家的硬币，拿出 *f* 个送给它的每一个相邻城市。例如，有个城市一天开始时拥有 *m* 个 *c* 类硬币，且有 *n* 个邻居，*f* 就是 *m*/1000 的整数部分，一天结束时总共要送出 *f**n* 个 *c* 类硬币。

数据范围：

（1）国家的个数 $1 \leqslant c \leqslant 20$。

（2）国家名称长度≤25。

（3）每个国家的顶点坐标是 $1 \leqslant x_l \leqslant x_h \leqslant 10$，$1 \leqslant y_l \leqslant y_h \leqslant 10$。

当某个城市中每种类型的硬币都至少出现了一个，就称这个城市“已经完成”。当一国的所有城市都已经完成时，就称这个国家已经完成。计算出每个国家都完成所需的天数。

【分析】

模拟的主过程就是一个循环，每次循环都遍历所有城市：

（1）将前一天收到的硬币加到余额中，然后判断是否已经完成。

（2）判断所在国家是否完成，如果所有国家完成，退出循环返回结果。

（3）按照指定的规则将所有城市的硬币向四周的城市扩散。

完整程序（C++11）如下：

```
using namespace std;

const int MAXC = 20 + 5, MAXX = 10 + 2, DX[] = {-1,1,0,0}, DY[] = {0,0,-1,1};
int c;

struct City{
   bool valid, complete; //是不是一个合法的城市，是否已经完成
```

```
    int motif[MAXC], in[MAXC], country; //各种硬币余额，入账的硬币个数，所在的国家
    void update(){
        assert(valid);
        _for(i, 0, c) if(motif[i] < 1) { complete = false; return; }
        complete = true;
    }
};
City cities[MAXX][MAXX];

struct Country {
    string name;                    //国名
    bool complete;                  //完成
    int xl, yl, xh, yh, day;        //边界，哪一天完成的
    void update(){
        _rep(x, xl, xh) _rep(y, yl, yh)
            if(!cities[x][y].complete) { complete = false; return; }
        complete = true;
    }

    bool operator<(const Country& rhs) const { //输出时的排序规则
        return day < rhs.day || (day == rhs.day && name < rhs.name);
    }
};
Country cts[MAXC];

void flowIn(int x, int y) {
    City& ci = cities[x][y];
    assert(ci.valid);
    _for(i, 0, c) ci.motif[i] += ci.in[i], ci.in[i] = 0;
    ci.update();
}

void flowOut(int x, int y) {
    City& ci = cities[x][y];
    assert(ci.valid);
    int n = 0;
    _for(i, 0, 4) if(cities[x+DX[i]][y+DY[i]].valid) n++;
    if(!n) return;

    _for(i, 0, c){
        int f = ci.motif[i] / 1000;
        ci.motif[i] -= f*n;
```



```
            _for(j, 0, 4){
                auto& cj = cities[x+DX[j]][y+DY[j]];
                if(cj.valid) cj.in[i] += f;
            }
        }
    }

    bool solve(int day) {
        bool ans = true;
        _for(ci, 0, c) {
            Country& ct = cts[ci];
            _rep(x, ct.xl, ct.xh) _rep(y, ct.yl, ct.yh) flowIn(x,y);
            ct.update();
            if(!ct.complete) ans = false;
             else if(ct.day == -1) ct.day = day;
        }

        if(ans) return true;

        _for(ci, 0, c){
            Country& ct = cts[ci];
            _rep(x, ct.xl, ct.xh) _rep(y, ct.yl, ct.yh) flowOut(x,y);
        }
        return false;
    }

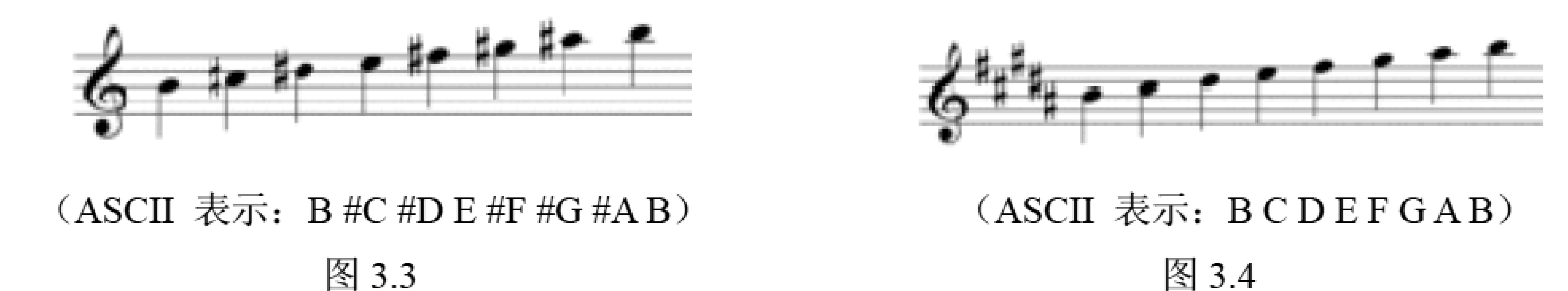
    int main(){
        int k = 1;
        while(cin>>c&&c) {
            cout<<"Case Number "<<k++<<endl;
            memset(cities, 0, sizeof(cities));
            _for(i, 0, c){
                Country& ct = cts[i];
                ct.complete = false; ct.day = -1;
                cin>>ct.name>>ct.xl>>ct.yl>>ct.xh>>ct.yh;
                _rep(x, ct.xl, ct.xh) _rep(y, ct.yl, ct.yh) {
                    City& ci = cities[x][y];
                    ci.valid = true, ci.country = i, ci.complete = false, ci.motif[i]
= 1000000;
                }
            }
```

```
        int d = 0;
        while(true) if(solve(d++)) break;
        sort(cts, cts+c);
        _for(i, 0, c) cout<<"   "<<cts[i].name<<"   "<<cts[i].day<<endl;
    }
    return 0;
}
```

优化调号（Optimizing Key Signature, UVa12568，难度 4）

看图 3.3 所示的乐谱，其调号为“0 #/b”（一般指的是 C 大调/A 小调）。

这个乐谱更合理的写法就是用“5#”作为调号，一般指的是 B 大调/g#小调。这种写法更自然，其中没有变音记号（升号#/降号 b/还原记号），如图 3.4 所示。



（ASCII 表示：B #C #D E #F #G #A B）
图 3.3

（ASCII 表示：B C D E F G A B）
图 3.4

给出一个带调号的乐谱片段，需要找到能够让变音记号（#/b）数目最小化的最优调号。注意你不能改变任何音符在五线谱上的位置。例如，即使能节省变音记号，也不能把调号#G 变成 bA。为简化考虑，只考虑“#”“b”和还原符号（没有升两个半音重升号或者升 1/4 音等），并且所有的音符都在同一个八度。不会有弧线跨越小节线连接两个音符。

将乐谱优化完成之后，对于音高已经由调号给出的音符，前面不能再附加任何变音记号，但是输入的乐谱中可能存在这些记号。注意：

- 在每个小节结束之前变音记号对特定的音符是一直起作用的，例如，音节 bA A A A 只包含一个变音记号，但是随后的 3 个音符 A 也受影响，所以 4 个 A 的音高相同。
- 变音记号是相对于还原音符而言的。例如，bG 是在还原 G 的基础上降低半音，比谱面的其他 G 低两个半音，因为其他 G 实际上是#G。也就是说，如果音符有其他升降号，是被覆盖而不是叠加。

给出形为 *m*#/b 的初始调号，表示调号中有 *m* 个“#”或者 *m* 个“b”（$0 \leqslant m \leqslant 7$）。如果 $m=0$，那么“#/b”部分就可以忽略。下面一行包含所有的由竖线“|”分隔，以双竖线“||”结束的乐谱片段。竖线和音符都由空格分隔开。每个音符包含两个字符，第一个是空字符、#、b 或者 n。第二个字符是大写的字母 C,D,E,F,G,A,B。输入中包含最多 10 个小节，100 个音符。

计算能够使乐谱中变音记号最少的调号，输出最小的记号个数以及最优的调号，格式和输入相同。调号只取 *m* 值最小的那个，然后按照字典序输出。

【分析】

按照原题中的附图，所有的调号及其作用的音符如表 3.1 所示。



表 3.1

1b	B	1#	F
2b	B E	2#	C F
3b	A B E	3#	C F G
4b	A B D E	4#	C D F G
5b	A B D E G	5#	A C D F G
6b	A B C D E G	6#	A C D E F G
7b	A B C D E F G	7#	A B C D E F G

为了方便处理，用取值范围为 0～6 的整数对应 A～G 的 7 个音符，这样就可用一个结构体来记录一个音节中的所有音符，这样方便处理每个音节中的变音记号的作用范围。

首先就是将输入的乐谱还原成调号为 0 时的表示方式，而且即使同一个音符在音节中前面有变音记号，后续的也不要省略。这样方便后续的处理，处理逻辑如下：对于每个音符，如果没有变音记号，首先看看小节前面是否出现过变音记号，如果出现过就用之前出现的。如果没有出现过，但是调号可以作用于这个音符，就使用调号的变音记号。扫描时可以维护一个当前每个音符在本小节内已经使用的变音记号的列表。

遍历所有可能的调号（注意也包括 0 调号），将调号应用到每个小节，检查结果的记号个数，每个小节内的处理逻辑如下。

遍历每个音符。如果音符没有记号或者记号为“n”，此时有两种情况。

（1）被当前的调号作用：如果音节中前面没有作用于这个音符的变音记号或者变音记号不是“n”，那么需要加一个变音记号。

（2）不被作用：如果前面有变音记号且不是“n”，那么需要加一个变音记号“n”。

如果音符记号为“#”或者“b”，也有两种情况。

（1）前面有变音记号：如果前面的变音记号和当前记号不同，那么需要加变音记号。

（2）前面没有变音记号，并且没有被调号覆盖，那么也需要加变音记号。

这样就可以算出每种调号下，所有音节需要的调号的个数。取最小值然后输出结果即可。

图像组合（Combining Images, World Finals 2003 - Beverly Hills, LA2726，难度 4）

因为要通过网络传输，图像压缩技术非常重要，它可以用相对少得多的位数来表示一个图像。有一种压缩算法叫“四分树”。如果一个图像的形状刚好是正方形，所有像素都是二进制（像素颜色是 0 或 1），组成一个边长都是 2 的幂的方阵，就可以用四分树来压缩，压缩方法如下：

（1）如果所有像素都是同色的，那么四分树的表示就是一个 1，后面跟着像素的颜色。例如，一个只包含颜色为 1 的像素的图形的四分树编码就是 11，与图像的大小无关。

（2）如果像素颜色不一，则四分树编码就是首先一个 0，后面依次跟着左上象限、右上象限、左下象限、右下象限的四分树编码。

四分树的二进制编码可以转换成十六进制表示。按照如下规则转换：

（1）首先在最左面加一个 1，作为分隔符。

（2）如果需要，在左边一直加 0 ，直到整个编码的长度为 4 的整数倍。

（3）依次把每 4 个比特转换成十六进制数的字符（0～15 依次对应 0～9 和 A～F）。

举例来说，仅包含颜色为 1 的像素的图形的编码就是 7(0111)。

给出两个图形的四分树十六进制表示，计算出两个图形的交集 A&B 并且输出其四分树十六进制表示。交集的规则如下：如果图形 A 和 B 的大小形状一致，A&B 的每个像素的颜色就是 A 和 B 对应位置像素颜色的“与”运算结果。

【分析】

关键的过程就是两个：

（1）递归解析树的二进制表示形式。这个只需要按照题目描述的过程编码即可。

（2）求两棵树的相交。

对于第二个过程，分 3 种情况：

（1）$p1$ 和 $p2$ 都是单色，则结果是单色的，颜色就是 $p1$ 和 $p2$ 颜色的与运算结果。

（2）其中之一（不妨假设为 $p1$）是单色的，如果 $p1$ 的颜色为 1，则相交结果就是 $p2$；否则就是 $p1$。

（3）如果 $p1$ 和 $p2$ 都不是单色的，则首先依次递归求 4 个子方阵的相交结果。然后再来判断，只有结果中 4 个子方阵都是单色，并且颜色一致，则结果才是单色的；否则不是单色的。

完整程序（C++11）如下：

```
using namespace std;s
int HEX[16][4] = { //十六进制每个字符对应的二进制
   {0,0,0,0},{0,0,0,1},{0,0,1,0},{0,0,1,1},{0,1,0,0},{0,1,0,1},{0,1,1,0},
{0,1,1,1},
   {1,0,0,0},{1,0,0,1},{1,0,1,0},{1,0,1,1},{1,1,0,0},{1,1,0,1},{1,1,1,0},
{1,1,1,1}
};

char HEXC[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E',
'F'};

const int MAXN = 2048;
typedef vector<int>::iterator IIter;

//HEX 转二进制
void fromHex(vector<int>& bits, const string& s) {
   bits.clear();
   int sz = s.size(), h;
   bool p = true, first = true;
   for(auto c : s){
```



```
        if (isdigit(c)) h = c - '0';
        else { assert(isupper(c)); h = c-'A'+10; }
        if(first) { //前 4 位的处理
            assert(h); first = false;
            _for(b, 0, 4){
                int x = HEX[h][b];
                if(p) { if(x) p = false; } //前缀 0 遇到 1
                else bits.push_back(x);
            }
        }
        else { _for(b, 0, 4) bits.push_back(HEX[h][b]); }
    }
}

string toHexStr(const vector<int>& bits) {
    int p = (bits.size() + 1) % 4, z = 1, i = 0, b = 0;
    string ans;
    if(p) { //需要附加前缀
        while(i < p-1) z = (z<<1) + bits[i++];
        ans += HEXC[z], z = 0;
    }
    else b = 1;

    while(i < bits.size()){
        assert(b < 4);
        z = (z<<1) + bits[i++];
        if(++b == 4) ans += HEXC[z], z = 0, b = 0;
    }

    return ans;
}

struct Node {
    int flag, v;            //是否为单色，单色的颜色
    Node *children[4];      //顺序的 4 个子方阵
    Node *init() { memset(children, 0, sizeof(children)); return this; }
};

MemPool<Node> nodes;
Node* newNode() { return nodes.createNew()->init(); }

Node* parseBits(const vector<int>& bits, int& b) {
```

```
    assert(b < bits.size());
    Node* p = newNode();
    p->flag = bits[b++];
    if(p->flag) { p->v = bits[b++]; return p; }
    _for(i, 0, 4) p->children[i] = parseBits(bits, b);
    return p;
}

Node* intersect(Node* p1, Node* p2) {
    assert(p1);
    assert(p2);
    if(p1->flag && p2->flag) {
        Node* p = newNode();
        p->flag = 1, p->v = (p1->v && p2->v);
        return p;
    }
    if(p2->flag) return intersect(p2, p1);
    if(p1->flag) return p1->v?p2:p1;

    Node* p = newNode();
    p->flag = 1;
    _for(i, 0, 4){
        assert(p1->children[i]); assert(p2->children[i]);
         Node *cp = intersect(p1->children[i], p2->children[i]);
        p->children[i] = cp;
         if(p->flag) {
             if(i == 0) p->v = cp->v;
             if(cp->flag == 0 || p->v != cp->v) p->flag = 0;
         }
    }
    if(p->flag) memset(p->children, 0, sizeof(p->children));
    return p;
}
void toBits(Node* p, vector<int>& b) { //四分树编码
    b.push_back(p->flag);
    if(p->flag) { b.push_back(p->v); return; }
    _for(i, 0, 4) toBits(p->children[i], b); //递归子树编码
}

int main(){
    string s1, s2;
    vector<int> bits;
```



```
    for(int t = 1; cin>>s1>>s2 && s1 != "0"; t++) {
        if(t>1) cout<<endl;
        fromHex(bits, s1);
        int b = 0;
        Node *p1 = parseBits(bits, b);
        fromHex(bits, s2); b = 0;
        Node *p2 = parseBits(bits, b);
        Node *p3 = intersect(p1, p2);
        bits.clear();
        toBits(p3, bits);
        cout<<"Image "<<t<<":"<<endl<<toHexStr(bits)<<endl;
        nodes.dispose();
    }
    return 0;
}
```

发现有竞争力的产品（Finding Competitive Products, ACM/ICPC Asia – Taichung 2014, LA7007，难度 5）

顶-k 查询非常有用。给出一个产品集合 D，其中的元素 p 用一个 d（$2≤d≤5$）维向量 $\{p[1],p[2],\cdots,p[d]\}$（$1≤p[i]≤512$）表示，其中 $p[i]$就是产品 p 的第 i 个特性值。一个顶-k 查询从 D 中获得一个大小为 k 的产品集合，查询首先使用一个用户偏好 $w=\{w[1], w[2], \cdots , w[d]\}$，其中 $w[i]$表示对于这个用户来说产品第 i 个特性的权值。查询时，使用一个打分函数对每个产品 p 进行打分：$f(p) = w[1]×p[1] + w[2]×p[2] +\cdots+ w[d]×p[d]$。对每个产品打分之后，就返回前 k（$1≤k≤35$）个分数最小的产品。注意，如果第 k 位有多个产品并列，都要加入查询结果。

现在定义热度 hot(p)为把产品 p 作为顶-k 查询结果的用户个数。给出客户偏好的集合 W，其大小$|W|$满足 $10≤|W|≤15000$，每个偏好包含 d 个整数（$0≤w[i]≤100$ $i = 1,2,\cdots,d$，不一定满足 $\sum w[i] =100$）。给出市场上现有产品的集合 EP，其大小$|EP|$满足（$5≤|EP|≤1500$），每个产品给出 d 个整数表示其特性值向量。

接下来是 t（$1≤t≤10$）个测试案例，每个案例包含一个整数 m（$5≤m≤15$）；接着是一个潜在产品的集合 PP，其大小$|PP|$满足 $5≤|PP|≤10000$。每个产品给出 d 个整数表示其特性值向量。对于每个测试案例，计算 PP 和 EP 在一起组成的集合中 m（$5≤m≤15$）个热值最高产品的热值之和。需要注意的是，有可能第 m 个位置有多个同样热值的产品，选其中任意一个计算即可。

【分析】

不难想到对于每个输入的产品的集合 PP，对于每个 W，计算所有 EP 和 PP 中产品的分数，然后进行排序，取最靠前的 k 个分数，然后对这些分数对应的产品热度递增。最后再按照题意对 PP 中的产品按照热度进行排序，取出前 m 个热度求和即可。

但是代码完成之后，大多数的读者应该会跟笔者一样，直接提交就是 TLE（程序超时）。

笔者经历了如下几次优化之后将代码的运行速度缩短到 15s 左右（题目要求是 60s 以内）：

（1）*EP* 和 *W* 集合都是固定的，那么 *EP* 中产品的分数就可以提前一次性计算好，而不用每次重复计算。

（2）要判断一个产品 *P* 是否是前 *k*，不需要把所有产品排序来取出前 *k* 个，在遍历所有产品计算分数时，维护前 *k* 个产品的分数以及对应的产品编号的列表即可。

（3）得到每个产品的热度之后，不用对所有的热度排序。我们只需要知道前 *m* 个最大的热度之和，可以使用 STL 中的 partial_sort 对所有的热度进行排序，其时间复杂度是 $n*\log(m)$，而本题中 *m* 比 *n* 小得多，而这样就又比 sort 节省了几倍的时间。

完整程序如下：

```
using namespace std;
const int MAXD = 8, MAXW = 15000 + 4,
   MAXEP = 1500 + 4, MAXPP = 10000 + 4, MAX_SCORE = 256004;
int d, t, k, wSize, epSize, ppSize, m, topCnts[MAXPP];
struct Prod{
   int p[MAXD];
   void readIn(){ _for(i, 0, d) scanf("%d", &(p[i])); }
   inline int operator*(const Prod& rhs){
      return inner_product(p, p+d, rhs.p, 0);
   }
};

Prod WS[MAXW], EP[MAXEP], PP[MAXPP];
vector<int> EP_Scores[MAXW];

int solve(){
   fill_n(topCnts, ppSize, 0);
   _for(wi, 0, wSize){
      vector<int> scores(EP_Scores[wi]), indice(scores.size(), -1);
      _for(pi, 0, ppSize){
         int ns = WS[wi]*PP[pi], maxS = scores.back();
         if(ns > maxS && scores.size() >= k) continue;
         if(ns >= maxS) {
            scores.push_back(ns), indice.push_back(pi);
            continue;
         }

         auto sit = lower_bound(scores.begin(), scores.end(), ns);
         indice.insert(indice.begin() + (sit-scores.begin()), pi);
         scores.insert(sit, ns);
         if(scores.size() >= k && scores[k] != scores[k-1])
            scores.resize(k), indice.resize(k);
```

```
        }

        _for(i, 0, scores.size()) if(indice[i] != -1) topCnts[indice[i]]++;
    }

    partial_sort(topCnts, topCnts + m, topCnts + ppSize, greater<int>());
    return accumulate(topCnts, topCnts + m, 0);
}

int main(){
    scanf("%d%d%d%d", &t, &d, &k, &wSize);
    _for(i, 0, wSize) WS[i].readIn();
    scanf("%d", &epSize);
    _for(i, 0, epSize) EP[i].readIn();
    _for(wi, 0, wSize){
        auto& mm = EP_Scores[wi];
        _for(ei, 0, epSize) mm.push_back(WS[wi]*EP[ei]);
        sort(mm.begin(), mm.end());
        if(mm.size() > k) mm.resize(k); //去掉肯定不是前 k 的 EP 产品
    }

    _for(i, 0, t) {
        scanf("%d%d", &m, &ppSize);
        _for(pi, 0, ppSize) PP[pi].readIn();
        printf("%d\n", solve());
    }
    return 0;
}
```

排版（Typesetting, World Finals 1994 Phoenix, LA5174，难度 5）

有一种比例字体，同一行文本中每一个字符大小可能不同，字符大小用“点数”来表示，而且整个文本的大小也影响其中每个字符的大小。给出一段文字，对其进行排版。文字中也可能包含指定随后字体以及文字点数的特殊单词。

输入字体宽度表，有 6 种不同的字体，同时给出了 N（$0 \leqslant N \leqslant 100$）个点数为 10 的字符在每一种字体中的宽度 w（$0 \leqslant w \leqslant 256$）。字符的宽度跟着点数可以线性缩放。例如 10 点的“A”宽度为 12 个单位，则 20 点的“A”宽度就是 24 个单位。注意，空格本身也是作为一种字符出现在这个表中。

给出一段文字，按照如下规则进行排版：

（1）文字包含 L 行，每一行的宽度限制为 W。

（2）每段文字的第一个字符都是 1 号字体，大小为 10 点。

（3）文字中的每个单词都是空格分隔的长度不超过 8 的字符串。

（4）单词中的每个字符都包含在输入的字体宽度表中。

（5）特殊的标记（如*f1、*f2、*f3、*f4、*f5 和*f6）用来给后续的文本指定字体编号。

（6）“*sN”（1≤N≤99）用来指定后续字符的点数。

（7）每一行在限制的宽度内，需要放下尽量多的单词，保证每个单词后面留出空格的宽度，空格的字体和字号跟其前面的字符一样。最后一个单词后面没有空格。

（8）如果一个单词排版出来的宽度超过 W，则独占一行。

（9）在对字体宽度进行缩放时，根据目标字号的宽度计算出来之后进行四舍五入处理。

【分析】

因为是对单词进行排版，首先想到的子过程是根据字体以及点数计算出一个单词的宽度。排版过程中需要维护以下几个变量：行号、字体、点数、当前行已经排版的宽度（不包含最后的一个空格）、最后一个空格的大小，以及当前行的所有单词。

对于每个单词，按照以下几种情况进行排版：

（1）指令单词（*f、*s），修改字体或者点数。

（2）新行的开头（lw=0），重置所有的全局变量。

（3）当前单词单独一行放不下，如果当前行已有单词，则换行。然后在新行中放入这个单词，之后另起一行。

（4）一行排版到一部分，剩余的空间放得下新单词以及前面的空格，则在当前行排版，否则另起一新行排版。

（5）所有单词排版完成之后，有可能一行空间还没用完，要作为独立的一行来考虑。

需要注意的是，单词之间空格的字号和大小是跟随其前面单词的设置。完整程序（C++11）如下：

```
using namespace std;

const int MAXN = 256 + 8, FCNT = 6;
const string SPACE = " ";
int Fonts[MAXN][FCNT+2];
typedef std::vector<string> StrVec;

int getWidth(int ft, int pt, const string& s) {
   int ans = 0;
   for(auto c : s) ans += (pt*Fonts[c][ft] + 5)/10;
   return ans;
}

void solve(const StrVec& words, int LW) {
   //行号，字体， 点数，已用的 LineWidth（不包含最后的一个空格），最后一个空格的大小
   int l = 1, ft = 1, pt = 10, lw = 0, lsw = getWidth(ft, pt, SPACE);
   StrVec lws;
   for(const auto& w : words) {
```

```
        assert(!w.empty());
        if(w[0] == '*') {
            if(w[1] == 'f') ft = w[2] - '0';
            if(w[1] == 's') sscanf(w.c_str(), "*s%d", &pt);
            continue;
        }

        int ww = getWidth(ft, pt, w);        //单词的宽度
        if(ww > LW) { //单词整体一行放不下
            if(lw) //当前不是新行，先换行
                printf("Line %d: %s ... %s (%d whitespace)\n",
                    l++, lws.front().c_str(), lws.back().c_str(), LW-lw);

            printf("Line %d: %s (%d whitespace)\n", l++, w.c_str(), LW-ww);
//放完这个单词另起一行
            lws.clear(); lw = 0;
            continue;
        }

        //新行
        if(!lw) {
            lws.push_back(w); lw = ww; lsw = getWidth(ft, pt, SPACE);
            continue;
        }

        //老行
        if(lw + lsw + ww > LW) { //放不下另起一行
            printf("Line %d: %s ... %s (%d whitespace)\n",
                l++, lws.front().c_str(), lws.back().c_str(), LW - lw);
            lws.clear(), lws.push_back(w), lsw = getWidth(ft, pt, SPACE), lw = ww;
            continue;
        }

        //放得下
        lws.push_back(w), lw += lsw + ww, lsw = getWidth(ft, pt, SPACE);
    }
    if(lw)
        printf("Line %d: %s ... %s (%d whitespace)\n",
            l++, lws.front().c_str(), lws.back().c_str(), LW - lw);
}

int main(){
```

```
    StrVec words;
    char line[128]; string word;
    bool first = true;
    int N, L, W;
    while(scanf("%d", &N) == 1 && N) {
        if(first) first = false; else puts("");
        memset(Fonts, 0, sizeof(Fonts));
        _for(i, 0, N-1) {
              scanf("%s", line);
            _rep(j, 1, FCNT) Fonts[line[0]][j] = readint();
        }
        _rep(j, 1, FCNT) Fonts[' '][j] = readint();

        for(int p = 1; scanf("%d%d", &L, &W) && L; p++){
            printf("Paragraph %d\n", p);
            gets(line);
            words.clear();
            _for(i, 0, L){
                gets(line);
                stringstream ss; ss<<line;
                while(ss>>word) words.push_back(word);
            }
            solve(words, W);
        }
    }
    return 0;
}
```

位图字体排版（Typesetting, ACM/ICPC North America - Mid Central - 2007/2008, LA3846，难度 5）

位图字体中，字形是用一系列的像素点来表示的，这种字体的排版方式称作字形压缩。压缩的规则是要让被压缩的两个字尽量接近，但是来自于两个字形的任意两个像素之间的水平间距至少为 1。

为了考虑到可能出现的垂直重合的情况（如图 3.5 所示），有些字体也设置了一些不可见的占位像素（如图 3.5 所示中的白点）。

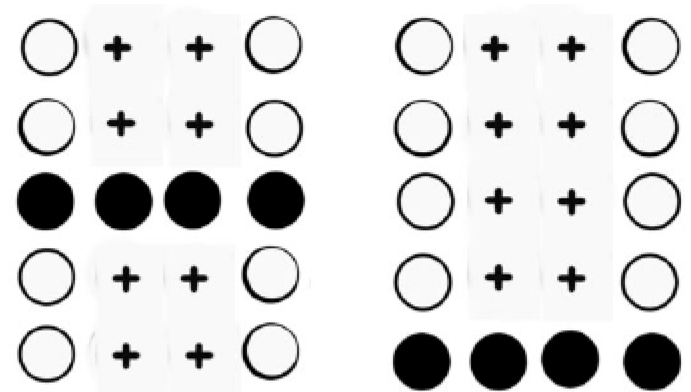
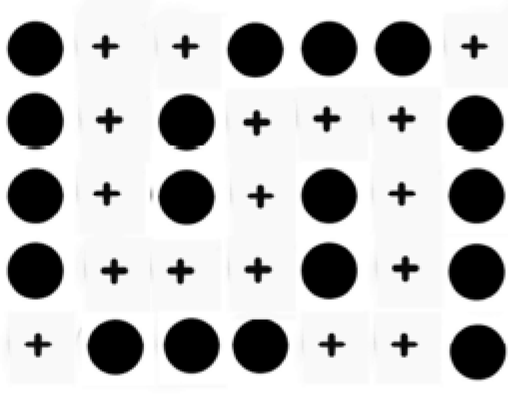


图 3.5



为了防止另外一种互相交叠的情况（如图 3.6 所示），需要增加另外一条规则，就是在同一条水平线上，左边字符的任意像素必须出现在右边字符的任意像素的左边。


图 3.6

给出多个行数相同的字形（大小不超过 20*20）的像素，输出最终的压缩结果。

【分析】

压缩规则虽复杂，但可以通过合适的建模进行处理：每一个字的表示中，除了像素之外，还要给出每一行的像素区间，也就是说最左边一个像素（包括不可见的那些）点和最右边像素点形成的区间。

两个字形在压缩时，首先要保证每一行的左字形的像素区间必须在右字形像素区间的左边，并且两个区间间距至少 1 个像素，这样不同的规则就可以统一处理。假设两个字形的宽度是 *wl* 和 *wr*，压缩的过程就是遍历右边字形第一列在最终压缩结果中相对于左边字形第一列的偏移量，这个偏移量可能在 $-wr$ 和 $wl+1$ 之间。找到最小的合法偏移量之后，就要更新最终压缩结果的字形宽度以及每一行对应的区间信息，以便进行下一个字形的处理。

完整程序（C++11）如下：

```
using namespace std;
const int MAXN = 24;
int N, gn;
struct glyph{
   string rows[MAXN];
   int segs[MAXN][2], width;          //每一行的像素区间，宽度
   void setRow(int ri, const string& row) {
      rows[ri] = row, width = row.size();
      setSeg(ri);
   }

   void setSeg(int ri) { //得到第 ri 行的区间信息
      int &l = segs[ri][0], &r = segs[ri][1];
      l = MAXN, r = -1;               //这一行的左右区间
      const string& row = rows[ri];
      _for(i, 0, width) if(row[i] != '.') l = min(l, i), r = max(r, i);
   }

   void pack(const glyph& rhs) {
      int c;
      for(c = -rhs.width; c <= width + 1; c++)
         if(canPut(c, rhs)) break;    //区间信息
      assert(c <= width + 1);
      if(c < 0) {
```

```
            int aw = -c;
            string pfx(aw, '.');
            width += aw;
            _for(i, 0, N) rows[i].insert(0, pfx), segs[i][0] += aw, segs[i][1]
+= aw;
            c = 0;
        }
        int nw = c + rhs.width;         //新的宽度
        if(nw > width) {
            _for(i, 0, N) rows[i].resize(nw, '.');
            width = nw;
        }

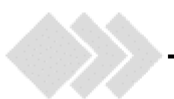
        _for(i, 0, N){
            _for(j, 0, rhs.width){
                char& cell = rows[i][j+c];
                if(cell == '.') cell = rhs.rows[i][j];
            }
            setSeg(i);
        }
    }

    bool canPut(int col, const glyph& rhs) { //是否会重叠
        _for(r, 0, N){
            int tl = segs[r][0], tr = segs[r][1],
                rl = rhs.segs[r][0], rr = rhs.segs[r][1];
            if(rr == -1 || tr == -1) continue;
            rl += col, rr += col;
            if(tr + 1 >= rl) return false;
        }

        return true;
    }

    void preOut() { //准备输出
        auto isColEmpty = [this](int col){ //第 col 列是否为空的，没字符
            _for(r, 0, N) if(rows[r][col] == '#') return false;
            return true;
        };

        for(int c = width - 1; c >= 0; c--) //压缩掉右边的空格
```



```
            if(isColEmpty(c)) --width; else break;

        int sj = 0;
        for(; sj < width; sj++) if(!isColEmpty(sj)) break; //左边的空格

        _for(i, 0, N) {
            string& r = rows[i];
            r.resize(width), r.erase(0, sj);
            for(auto& rc : r) if(rc == '0') rc = '.';
        }
    }
};

ostream& operator<<(ostream& os, const glyph& g) {
    _for(i, 0, N) os<<g.rows[i]<<endl;
    return os;
}
int main(){
    string line,r;
    glyph gs[MAXN];
    for(int t = 1; cin>>N && N; t++) {
        getline(cin, line);
        cout<<t<<endl;
        _for(i, 0, N) {
            getline(cin, line);
            stringstream ss(line);
            gn = 0;
            while(ss>>r) gs[gn++].setRow(i, r);
        }
        glyph& g0 = gs[0];
        _for(i, 1, gn) g0.pack(gs[i]);
        g0.preOut();
        cout<<g0;
    }
}
```

城市道路（City Directions, UVa163，难度 6）

城市中大道是南北向，大街是东西向，干道是对角向。最中间大道和大街标记为 0(A0, S0)。其他的道路依次命名，A3W 是 A0 西面的第 3 条大道。总共有 6 条干道，有两个穿过市中心，每个象限内还有一个。图 3.7 显示了较小版本的这种城市的西北象限。在干道穿过的路口，可以做 45°（half）或者 135°（sharp）转向。

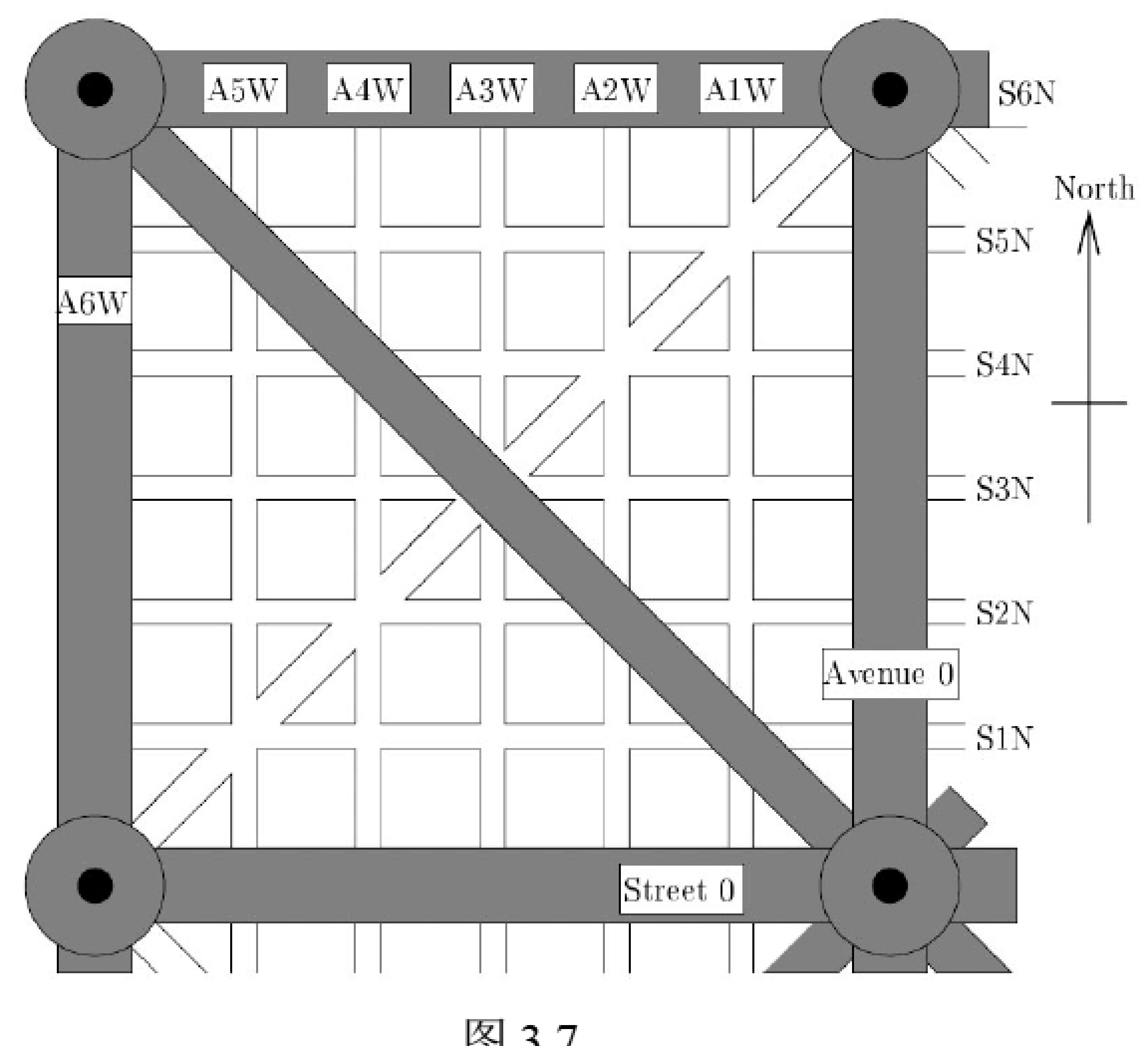


图 3.7

标记为灰色的道路称为快速路。它们只能在圆圈那里交叉，交叉路口也被经过的其他道路共享。只能通过左转（对于干道来说是 135° 左转）进入。快速路上不能停车，其他的道路则没有这些限制。

车在这个路网中的位置使用最后经过的交叉路口和当前的方向来确定，方向用北（N）、东北（NE）、东（E）、东南（SE）、南（S）、西南（SW）、西（W）和西北（NW）来表示。行驶的指令也通过经过的路口个数和转弯的种类来给出。指令应该遵循以下语法（可能会输入错误的语法）：

（1）命令 ::= 转弯命令 | 执行命令。

（2）转弯命令 ::= TURN [HALF|SHARP] {LEFT|RIGHT}：执行对应的转弯。

（3）直行命令 = GO [STRAIGHT]：n 直行经过 n（$0 \leqslant n \leqslant 99$）个路口。

本题中的城市，每个象限是 50×50 个街区，整个城市是 100×100，最外层的快速路编号中的数字为 50，并且大小干道在编号 25 的道路交叉。输入你的起始位置和方向，然后给出一系列指令。如果指令语法有错，或者会导致非法或者无效的转弯，就忽略它。无论何时，指令都不会把你带出城市边界。每个测试案例都以一条“STOP”指令来结束。即使起始位置是正中间的道路(A0,S0)，也会附上 N 或 E。

输出停下之后位置，如果此位置非法，输出“Illegal stopping place”。

【分析】

这个题目的模型比较复杂，这里可以引入几个二维几何以及图论的结构来简化处理过程。可以把交叉路口看成结点，然后把道路看成边，使用邻接矩阵存储，然后就开始模拟汽车的行驶过程。

在指令执行的过程中维护当前位置、当前方向，以及按照当前方向走到的下一个位置，这样方便判断拐弯是否有效，以及是否违反了进入快速路的转弯规则。

需要注意的是，题目输入的开始中有可能出现连续两条设置位置的指令，只需考虑第一条即可。完整程序（C++11）如下：



```
using namespace std;

bool eq(const char *l, const char *r) { return !strcmp(l, r); }

struct Point{
   int x, y;
   Point(int a=0, int b=0) : x(a), y(b) {}
};
typedef Point Vector;
Vector operator * (const Vector& A, int n) { return Vector(A.x*n, A.y*n); }
bool operator == (const Point& A, const Vector& B) { return A.x == B.x &&
A.y == B.y; }
Point operator + (const Point& A, const Vector& B) { return Point(A.x+B.x,
A.y+B.y); }

struct Line {
   int a, b, c; //a*x + b*y = c
   Line(int A, int B, int C) : a(A), b(B), c(C) {}
   bool onLine(const Point& A) const { return A.x*a + A.y*b == c; }
};

vector<Line> throughWays = {
   Line(1,0,0), Line(0,1,0), Line(1,-1,0), Line(1,1,0),
   Line(1,0,50), Line(1,0,-50), Line(0,1,-50), Line(0,1,50) };
const Vector N(0,1), NE(1,1), E(1,0), SE(1,-1), S(0,-1), SW(-1,-1), W(-1,0),
NW(-1,1);
vector<Vector> dirs = {E, NE, N, NW, W, SW, S, SE};
vector<string> dirNames = {"E", "NE", "N", "NW", "W", "SW", "S", "SE"};
map<string, Vector> dirNameMap;
const int SIZE = 50;
struct Graph {
   set<int> g[10201];

   Graph() { //建图
      _for(x, -SIZE, SIZE+1) _for(y, -SIZE, SIZE+1) { //水平和垂直间隔之间相连
         connect(Point(x,y), Point(x, y+1));
         connect(Point(x,y), Point(x+1, y));
      }

      _for(x, -SIZE, SIZE) {
         connect(Point(x, x), Point(x+1, x+1));
         connect(Point(x, -x), Point(x+1, -x-1));
```

```
            }

            _for(x, 0, SIZE) {
                int y = SIZE - x, x1 = x + 1, y1 = SIZE - x1;
                connect(Point(x, y), Point(x1, y1));
                connect(Point(-x, -y), Point(-x1, -y1));
                connect(Point(-x, y), Point(-x1, y1));
                connect(Point(x, -y), Point(x1, -y1));
            }
        }

        inline void connect(const Point& A, const Point& B) {
            assert(valid(A));
            if(!valid(B)) return;
            int a = toInt(A), b = toInt(B);
            g[a].insert(b), g[b].insert(a);
        }

        inline int toInt(const Point& A) { return (A.x + SIZE) * SIZE * 2 + A.y
+ SIZE; }

        inline bool valid(const Point& A) { return -SIZE <= A.x && A.x <= SIZE
&& -SIZE <= A.y && A.y <= SIZE; }

        bool connected(const Point& A, const Point& B) {
            if(!valid(A) || !valid(B)) return false;
            return g[toInt(A)].count(toInt(B));
        }
    };

    bool locSet;
    Point pos, headPos;
    int dir;
    Graph graph;

    void printLoc(const Point& p) {
        printf("A%d", abs(p.x));
        putchar(p.x>=0 ? 'E' : 'W');

        printf(" S%d", abs(p.y));
        putchar(p.y>=0 ? 'N' : 'S');
        printf(" %s\n", dirNames[dir].c_str());
```



```
}

bool onThroughWay(const Point& p, int d) {
    const Vector& dv = dirs[d];
    Point p2 = p + dv;
    for(auto l : throughWays) if(l.onLine(p) && l.onLine(p2)) return true;
    return false;
}

bool posValid(const Point& p) {
    int ret = -SIZE <= p.x && p.x <= SIZE && -SIZE <= p.y && p.y <= SIZE;
    //if(!ret)
        //printf("%d, %d", p.x, p.y);
    return ret;
}

//d : 顺时针（-）或者逆时针（+）转多少个 45°
bool tryTurn(int d, Point& newPos, Point& newHeadPos, int& newDir) {
    newDir = (dir + d + 8) % 8;
    const Vector& dv = dirs[dir], ndv = dirs[newDir];
    newPos = pos + dv;                //先继续走一个路口
    newHeadPos = newPos + ndv;        //按照新的方向设定前方的下一个路口
    assert(newPos == headPos);
    int nx = newPos.x, ny = newPos.y;

    if(!posValid(newPos)) return false;

    //首先检查拐弯是否有效（在你要拐到的方向上是有路的）
    if(posValid(newHeadPos) && !graph.connected(newPos, newHeadPos))
        return false;

    //如果是在 9 个特殊的环岛路那里，那么任意方向拐弯都是有效的
    if((!nx && !ny) ||
        (abs(nx) == 0 && abs(ny) == SIZE) ||
        (abs(ny) == 0 && abs(nx) == SIZE) ||
        (abs(ny) == SIZE && abs(nx) == SIZE))
        return true;

    //进入一个高架
    if(onThroughWay(newPos, newDir) && !onThroughWay(pos, dir)) {
        if(fabs(ndv.x) == fabs(ndv.y)) {
            if(d != 3) return false;
```

```
        }
        else if(d != 2) return false;
    }

    //离开一个高架
    if(onThroughWay(pos, dir) && !onThroughWay(newPos, newDir)) {
        if(fabs(dv.x) == fabs(dv.y)){
            if(d != 3) return false;
        }
        else if(d != 2) return false;
    }

    return true;
}

bool cmdValid(const char *cmd, vector<string>& parts) {
    char p1[16] = {0}, p2[16] = {0}, p3[16] = {0};
    int ret = sscanf(cmd, "%s%s%s", p1, p2, p3);
    if(ret == 2) {
        parts.emplace_back(p1), parts.emplace_back(p2);
        if(p1[0] == 'G') {
            int l = strtoul(p2, NULL, 0);
            return eq(p1, "GO") && l >= 1 && l <= 99;
        }
        return eq(p1, "TURN") && (eq(p2, "RIGHT") || eq(p2, "LEFT"));
    }
    else if(ret == 3) {
      parts.emplace_back(p1), parts.emplace_back(p2), parts.emplace_back(p3);

        if(p1[0] == 'G') {
            int l = strtoul(p3, NULL, 0);
            return eq(p1, "GO") && eq(p2, "STRAIGHT") && l >= 1 && l <= 99;
        }

        return eq(p1, "TURN") && (eq(p2, "HALF") || eq(p2, "SHARP"))
            && (eq(p3, "RIGHT") || eq(p3, "LEFT"));
    }

    return false;
}

void exec(const char *cmd) {
```



```
        if(cmd[0] == 'A') {
            if(locSet) return;
            char p1[16], p2[16], p3[16];
            sscanf(cmd, "%s%s%s", p1, p2, p3);
            char xc, yc; int x, y;
            sscanf(p1+1, "%d%c", &x, &xc);
            sscanf(p2+1, "%d%c", &y, &yc);
            if(xc == 'W') x = -x;
            if(yc == 'S') y = -y;
            pos.x = x, pos.y = y;
            dir = find(begin(dirs), end(dirs), dirNameMap[string(p3)]) - begin
(dirs);
            headPos = pos + dirs[dir];
            assert(graph.connected(pos, headPos));
            locSet = true;

            //printLoc();
            return;
        }

        vector<string> parts;
        if(!cmdValid(cmd, parts)) {
            //printf("%s - invalid ignore\n", cmd);
            return;
        }

        if(cmd[0] == 'T') {
            int d = 2;
            if(parts.size() == 3) {
                if(parts[1][0] == 'H') d--;
                else if(parts[1][0] == 'S') d++;
                else assert(false);
            }
            if(parts.back()[0] == 'R') //RIGHT
                d = -d;

            Point newPos = pos, newHeadPos;
            int newDir = dir;

            if(tryTurn(d, newPos, newHeadPos, newDir)) {
                pos = newPos;
                headPos = newHeadPos;
```

```
            dir = newDir;
        }
    } else if(cmd[0] == 'G') {
        int l = strtoul(parts.back().c_str(), NULL, 0);
        pos = pos + dirs[dir] * l;
        assert(posValid(pos));
        headPos = headPos + dirs[dir] * l;
    }
}

void stop() {
    if(onThroughWay(pos, dir)) puts("Illegal stopping place");
    else printLoc(pos);
}

int main(){
    _for(i, 0, 8) dirNameMap[dirNames[i]] = dirs[i];
    locSet = false;
    char buf[256];
    while(true) {
        gets(buf);
        if(!strlen(buf)) continue;
        if(eq(buf, "STOP")) stop(), locSet = false;
        else if(eq(buf, "END")) break;
        exec(buf);
    }

    return 0;
}
```

忒修斯和米诺斯（Theseus and the Minotaur (II), World Finals1995 Nashville, LA5182，难度 6）

有一个迷宫，由山洞以及连接山洞的隧道组成。有两个角色 T（忒修斯）和 M（米诺斯），开始都在某个隧道里，然后开始向前走。

T 的运动规则是：

（1）进入到一个洞穴，靠右走。

（2）遇见隧道口，如果这个隧道没走过，他就标记一下，然后沿着它向前走。

（3）如果标记过，他就寻找逆时针顺序的下一个隧道口。

M 的运动规则是相反的：靠左侧走。也就是顺时针地选取隧道口，标记则跳过，没标记就标记，然后走进去。

如果他们在同一山洞相遇，T 杀死 M；如果他们在隧道相遇，M 杀死 T。如果 T 走进



的山洞，发现 M 在他之前走过，那么他就会在这个山洞中点燃一根蜡烛，然后沿着 M 最后走过的隧道前进。如果 M 走进山洞时，发现有蜡烛，他就退回到上一个山洞。

计算出最后谁杀了谁？

考虑图 3.8：

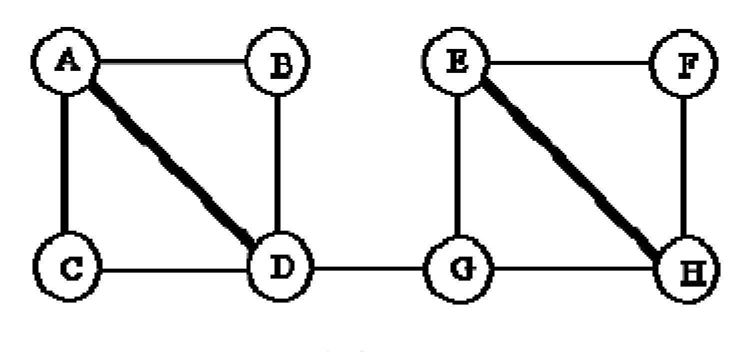


图 3.8

假设 T 在隧道 AC 中开始朝 C 走，M 在 FH 内朝 H 走。T 进入 C 之后，逆时针方向朝 D 走，同时 M 在进入 H 之后朝 G 走。然后 T 往 G 方向走，M 也朝 D 走。然后在隧道 DG 内，T 就被 M 杀掉。

反过来说，如果 T 起点不变，而 M 在 DG 之间开始，当 T 本来要以 C→D→G 这样移动时，M 是 G→E→F。当 T 进到 G 里面时，就可以看到 M 之前已经来过，所以朝 E 而不是 H 走了。然后在 M 到达 H 时到达 E。此后 M 再从 H 到 G 之后发现 G 已经点了蜡烛，就回头了，然后和 T 同时到达 H，这样 M 就被 T 杀掉。这个过程中 T 和 M 的状态变化如表 3.2 所示。其中，两个字母 AC 表示正在 A→C 之间的隧道中。

表 3.2

T	M
AC→C	DG→G
C	G
CD→D	GE→E
D	E
DG	EF
G（发现 M 来过并且最后一次去了 E）	F
GE	FH
E	H
EF	HG
F	G
FH	GH（发现 G 里面有蜡烛，所以返回 H）
H	H

【分析】

题意理解非常重要，尤其以下几点：

（1）T 和 M 是分别做标记，互相没有影响。

（2）标记都是在洞中做给各个隧道口的，而不是隧道本身的，所以一个隧道的两端有两个标记。

（3）M 在发现隧道前方的洞中有蜡烛时，不会进去（也就不会在其中被 T 杀掉），而是直接在隧道中掉头返回之前的洞中。

明确题意之后，定义洞穴对应的结构：

```
struct Node{ //洞穴
    bool candle, TVis[MAXN], MVis[MAXN]; //蜡烛，T 和 M 给通往其他洞穴的隧道口标记
    vector<int> adjs;                    //邻居洞穴的编号，逆时针顺序
```

```
    int id;            //洞穴的编号，'A'～'Z'分别是 0~26
int lastMTarget;       //M 来过之后，最后去哪个洞穴
    Node() : candle(false), lastMTarget(-1) {
        fill_n(TVis, MAXN, false);
        fill_n(MVis, MAXN, false);
    }
};
```

模拟过程其实就是一个循环，每一步包含以下步骤：

（1）M 准备进洞，但是如果发现前方有蜡烛，直接扭头返回来源的洞中。

（2）T 进洞，如果发现 M 在洞中，杀之。如果发现 M 的踪迹，点蜡烛。

（3）T 如果在洞中发现 M 的踪迹，跟着进去；否则逆时针寻找下一个出口，标记之后进去。

（4）M 在洞中顺时针寻找下一个出口，首先标记，然后在洞中记录 M 的踪迹之后进去，如果在隧道中发现 T，杀之。

窗口框架（Window Frames, World Finals - San Jose 1997, UVa513，难度 6）

有一种图形界面系统，使用称为框架的特殊矩形来表示各种窗口以及控件。如果一个框架内部的全部或者部分空间分配给了其他的框架，就称这个框架为父框架，内部的框架称为子框架。没有父亲的框架称为根，它的大小由用户指定。现在需要你来确定放置在根框架内部的多个框架的大小和位置。

框架中的空洞就是其中还没被子框架占用的空间。当一个新的子框架创建之后，会在空洞的顶端或者底端分配一个水平条形的空间，称为水平子框架。在左右两个边缘分配的垂直条状框架称为垂直子框架。创建一个新的子框架之后，空洞变小，但是其形状依然是矩形。在外围框架中放置子框架的过程称为打包。子框架依据打包的顺序依次在空洞中布局。

图 3.9 中，框架 1 先打包，接着是底边上的 2，左边的 3，最后是右边的 4。白色区域表示空洞，包含用来打包后续子框架的可用空间。

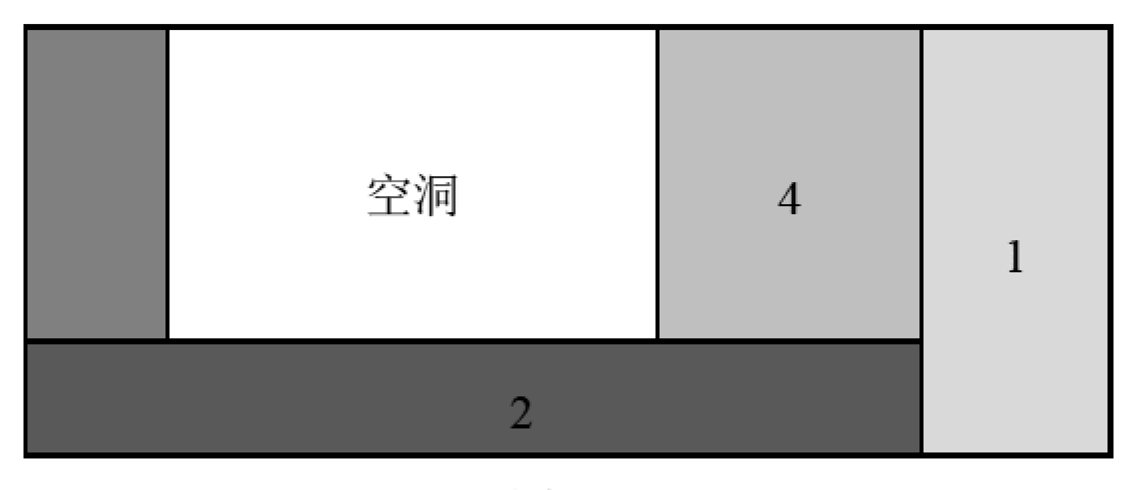


图 3.9

每个框架都是像素组成的矩形网格。如果根框架占用 r 行 c 列像素，那么左上角的坐标就是(0,0)，而右下角是$(c-1,r-1)$。框架的位置由其左上角和右下角的像素点坐标确定。

每个框架都有一个最小尺寸，由一个输入参数 d 及其子框架的最小尺寸决定。一个框架必须足够大才能打包所有的孩子。每个框架的最小尺寸由如表 3.3 所示规则确定。



表 3.3

打包边缘	框架类型	最小宽度	最小高度
左右	垂直	max（*d*,子框架所需宽度）	max（1,子框架所需高度）
上下	水平	max（1,子框架所需宽度）	max（*d*,子框架所需高度）

如果一个框架变得比上文所说的最小尺寸更大，多余的空间就分配给它的子框架以及空洞部分。每个框架输入有一个扩展标志，设置之后就表示垂直框架可以变宽或者水平框架可以变高。例如，一个设置了扩展标志的水平框架，之前是在空洞的顶端分配的空间，现在就可以向下扩展变得更高。

框架中增加的水平空间是这样分配的：记 x 为父框架超过其最小宽度的部分，n 为其设置了扩展标志的垂直子框架个数。那么 x 像素就在 n 个子框架中平均分配。如果 q 是 x 除以 n 的商，r 是余数。那么 n 个垂直框架都增加 q 像素的宽度，而且其中前 r 个除了 q 之外再增加宽度 1。如果 n=0，那么垂直框架宽度不变，x 个像素全部增加到空洞中。无论如何，需要时水平子框架要变宽来保证空洞的矩形形状。

增加垂直空间的分配方法类似，只是空间的增长方向不同。设置了扩展标志的水平子框架变高。如果没有，垂直空间的高度做相应的增加。同时，需要时垂直子框架变高以保证空洞的矩形形状。

在图 3.10 中，左边的根框架扩大成右边的形状。6 和 7 是 5 的垂直子框架，而且只有 4、6、7 设置了扩展标志。在右图中，增加的水平和垂直空间已经被分配到子框架中，结果是图中箭头所示的尺寸增长。注意框架 6、7 大小都不变，因为其父框架 5 中没有空间用来扩展。

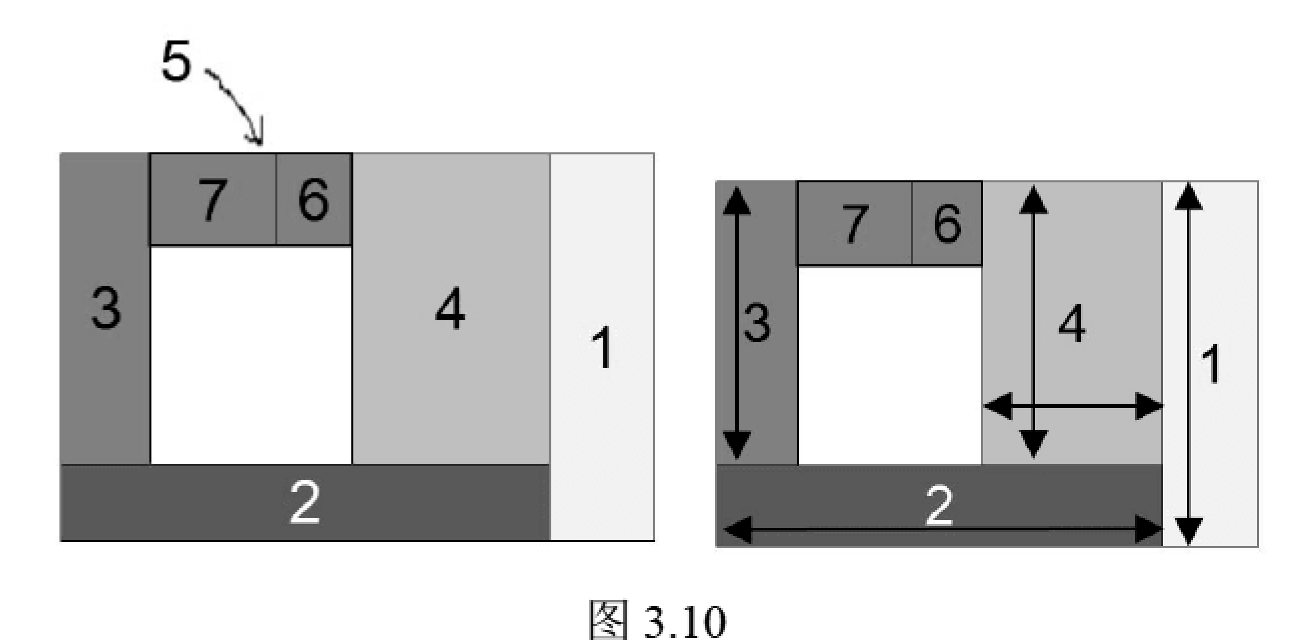


图 3.10

输入一系列的根框架和其子孙，以及根框架可能的不同大小。每个根框架给出除了根之外的子框架个数 M，以及根大小的不同个数 N，二者都是正整数。

接下来的 M 行每一行给出一个框架的信息：n p s d e，其中：

（1）n 是框架的名称（正整数）。

（2）p 是其父框架的名称（0 表示根）。

（3）s 是如下 4 种字符“L”“R”“T”“B”之一，表示打包方向，分别对应左、右、上、下。

（4）d 是最小尺寸（正整数）。

（5）e 是 0 或者 1，扩展标志。

接下来 N 行，每一行给出两个正整数 c、r，分别是根框架的像素行列数。根框架没有

列出，同一个根下面的不同的框架编号不同。子框架不会在其父框架前输入。框架按照输入的顺序进行打包。M 和 N 为 0 表示输入结束。

对于根框架的每一个输入的大小，输出其最终大小并且列出每一个子框架的名称机器左上角和右下角坐标。按照在父框架中打包的顺序给出，首先是根框架的第一个子以及子孙，接着是第二个子以及其孙……如果根框架的体积不足以打包，输出“is too small”。

【分析】

首先是定义一个结构来表示 Frame：

```
struct Frame{
int n, p, d, e;
int minW, minH, W, H;          //计算出的最小宽高，缩放的目标宽高
   char s;                     //打包方向
   bool isHori, isVert;        //水平垂直的标志
   vector<Frame*> children;    //子 Frame

   Frame() : n(0), d(1), isVert(true), isHori(false), s('r') { }
   void calcMinSize();         //计算最小的宽高
   void distribute() ;         //将多余的宽高分配到各个子 Frame 中
   void print(ostream& os, int lR, int lC, int rR, int rC);
//在指定的区域中输出结果
};
```

输入时根据 s 设置水平和垂直标志。同时维护一个 map<int, Frame*>记录 n 到 Frame 的对应关系。首先是加入 root Frame，之后每输入一个 Frame 都从这个 map 中查找到父 Frame，并加入父 Frame 的 children 结构中。

之后递归计算每个 Frame 所需要的 minW 和 minH。过程如下：根据 Frame 的水平或者垂直类型，以及 d，设置初始的 minW 和 minH，并且设置对应的空白的宽高 cavW 和 cavH。

之后按照顺序遍历每一个 Frame：

（1）先递归计算子 Frame 的 minW 和 minH。

（2）如果发现空白的宽高不够放这个 Frame，那么就先增加 minW 和 minH，差多少，增加多少。空洞的宽高也要设置成子 Frame 的宽高。

（3）cavW 和 cavH 都减去子 Frame 的 minW 和 minH。

完成上述计算之后，首先比较一下 root 的 minW、minH 和输入的 R、C，如果大小不够，直接输出“is too small”即可。

按照题目描述的规则递归地将多余的宽高分配到子 Frame 中去：

（1）计算各个子 Frame 的新的宽高。

（2）将空白设置成新的宽高，重新根据子 Frame 的宽高以及计算出来的 cavW、cavH 来分别设置水平子 frame 的宽以及垂直子 frame 的高。

（3）递归对每个子 frame 调用分配宽高的过程。

分配完宽高之后，就需要输出。如果要在每个 Frame 中记录坐标就比较麻烦，其实可



以使用布局分配类似的逻辑来递归输出坐标：

（1）输出自身的坐标，挨个遍历子 Frame，根据其宽高，计算出坐标，递归输出。

（2）再根据子 Frame 宽高，改变剩余区域的坐标，供后续子 Frame 使用。

绿鸡蛋和火腿（UVa10155 - Green Eggs and Ham，难度 7）

给出包含加（+）、减（-）、乘（*）、除（/）、指数（^）和等号（=）的表达式。注意，“^”是自右向左集合，其他运算符是自左向右集合。运算符优先级如下：

（1）一元的负号（-）。

（2）^。

（3）*和/。

（4）+和-。

（5）=。

括号运算符()（显式）和{}（隐式）用来修改运算符。输入包含以上运算符，实数（12、1.2、.35 等）以及单字母的变量的表达式。每个表达式语法都是正确的，且独占一行。实数的格式是包含一到多个十进制数字且最多一个小数点的字符串，实数不会包含指数。

按照如下规则输出表达式：表达式的每个部分占用一个矩形盒子，并且包含一个逻辑垂直中心线（LVC）。表达式以各种方式将矩形盒子连接起来。数字和变量的盒子宽度和其字符串相同（44.4 占 4 个字符宽），高度也是 1 个字符，其 LVC 就是包含其文本的那一行。

在表达式 E 前面附加一个一元负号运算符形成的表达式比 E 宽 1 个字符。LVC 和 E 相同，并且字符“-”就会出现在 E 的 LVC 的左边。

除了“/”和“^”之外的二元运算符连接两个表达式 E1 和 E2，结果中两个表达式相距 3 个字符，运算符出现在二者 LVC 的正中间，这样就确定了新表达式的 LVC。

两个表达式 E1 和 E2 形成的除法表达式中，结果中会是 E1，横线“-”组成的水平线，E2 三者垂直摞起来。水平线的长度等于 E1 和 E2 中最长的那个的宽度。二者之间更短的那个要水平居中，如果不能严格水平居中，那么就在右边加一个空格。水平线形成结果的 LVC。

指数操作符“^”把底数表达式和指数表达式连起来。指数表达式在底数的右上方，不增加任何空格，结果的 LVC 就是底数的 LVC。

隐式括号“{}”不改变它包含的表达式的 LVC 的显示结果。显式括号在被包含的表达式的 LVC 的左右两边增加 l 两列“(”和“)”。

样例输入：

```
x^n+y^n=z^n
1/{1+1/{1+1/{1+x}}}
123/1/12
123/{-1/12}
a^b^c+4/(1+x/{1-x})^{x-y}
```

样例输出：

```
n    n    n
```

```
x  + y  = z

     1
-------------
       1
1 + ---------
         1
   1 + -----
       1 + x

123
---
 1
---
 12

123
---
 -1
 --
 12

  c
 b           4
a   + ----------------
                x - y
     (      x  )
     (1 + -----)
     (    1 - x)

-1 + 2
```

【分析】

首先需要进行词法分析，将输入语句分成独立的单词，除了数字之外的所有单词都是单字符，只需对数字进行特殊处理即可。

词法分析完成之后，使用之前介绍过的递归下降法构建表达式树。需要注意的是，表达式树的每一个结点都要记录是否有负号以及被显式括号“()”包围。如果一个表达式被括号包括，就要把括号这一层记录成一个包含一个结点的树枝，以处理一个表达式被多层括号包括的情况。在遇到减号“-”时，需要单独判断如果上一个单词是表达式，说明这个减号是一个一元运算符。

构造完表达式树之后，就可以从树根往下递归地构造字符矩阵，用如下结构来表示：

```
struct Box {
    int LVC, w, h;                                          //LVC，宽度，高度
    vector<string> grid;                                    //字符矩阵
    void setGrid(int width, int height) {}                  //设置成指定的宽和高
    void setGrid(bool neg, const string& op) {}             //设置成指定的操作数字符串，
                                                            /* 同时指定是否有负号 */
    void addGroup(bool group, bool neg){}               //加上括号，同时指定是否有负号
    void copyTo(Box& b, int r, int c) {}                //将当前的字符矩阵复制到另外
                                                        /* 一个矩阵的指定位置 */
};
```

对于指定的树结点 p：

（1）如果 p 不是操作符，构造一个字符矩阵，就是单行的字符串。

（2）如果 p 是括号结点，首先递归构造子结点的矩阵，然后在矩阵周围套一层括号和负号。

（3）如果 p 的操作符是“^”，将左右两个子结点的矩阵以左下、右上的布局叠加起来即可。

（4）如果 p 的操作符是“/”，将两个子矩阵上下叠加，中间加一根线即可。

（5）对于其他操作符，构造一个新的矩阵，中间加上一个操作符即可，高度就按照题目指定的规则来构造。

需要注意的是，以上操作都会复用 Box.copyTo 操作，将当前的字符矩阵复制到结果矩阵的指定区域。

完整程序（C++11）如下：

```
using namespace std;
typedef vector<string> SVec;

struct Box {
    int LVC, w, h;
    SVec grid;

    void setGrid(int width, int height) {
        assert(width > 0); assert(height > 0);
        grid.clear();
        w = width, h = height;
        grid.resize(h);
        _for(i, 0, h) grid[i].resize(w, ' ');
    }

    void setGrid(bool neg, const string& op) { //设置操作符
```

```
        h = 1, LVC = 0;
        grid.clear(), grid.push_back(op);
        if (neg) grid[0].insert(0, "-");
        w = grid[0].size();
    }

    void addGroup(bool group, bool neg){ //加括号
        _for(i, 0, h) {
            string& l = grid[i];
            if (neg) {
                if (i == LVC) l.insert(0, "-");
                else l.insert(0, " ");
            }
            if (group) {
                l.insert(0, "(");
                l.push_back(')');
            }
        }
        if (group) w += 2;
        if (neg) w += 1;
    }

    void copyTo(Box& b, int r, int c) {
        _for(i, 0, h) _for(j, 0, w) {
            assert(i + r < b.h); assert(j + c < b.w);
            b.grid[r+i][c+j] = grid[i][j];
        }
    }
};

ostream& operator<<(ostream& os, Box* p) {
    _for(i, 0, p->h) {
        assert(p->w == p->grid[i].size());
        os<<p->grid[i]<<endl;
    }
    return os;
}

struct Node {
    string op;
    bool inGroup, neg;
    Node *left, *right;
```

```
    Node() : inGroup(false), neg(false), left(NULL), right(NULL) { }
};

MemPool<Node> nodesPool;
MemPool<Box> boxPool;
map<string, int> ops{{"=",0},{"+",1},{"-",1},{"*",2},{"/",2},{"^", 3}};
//运算符优先级
void tokenize(const string& s, SVec& ts) { //词法分析
    string buf; ts.clear();
    for(auto c : s){
        if (isdigit(c) || c == '.') { buf += c; continue; }
        if (!buf.empty()) ts.push_back(buf);
        buf.clear();
        ts.push_back(string(1, c));
    }
    if (!buf.empty()) ts.push_back(buf);
}

bool isOp(const string& t) {
    assert(!t.empty());
    return !(t[0] == '.' || isalnum(t[0]));
}

Node* buildExpTree(const SVec& ts, int l, int r) {
    Node* p = nodesPool.createNew();
    assert(l <= r);
    if (l == r) {
        assert(ts[l][0] == '.' || isalnum(ts[l][0])); p->op = ts[l];
    }
    else if (l + 1 == r) {
        assert(ts[l] == "-"); p->op = ts[l + 1]; p->neg = true;
    }
    else {
        int i0 = -1, i1 = -1, i2 = -1, i3 = -1, dep = 0;
        string lt = "_";
        _for(i, l, r + 1) {
            const string& t = ts[i];
            if (t == "{" || t == "(") dep++;
            else if (t == "}" || t == ")") dep--;
            if (dep) continue;
            if (t == "=") i0 = i;
            else if (t == "+") i1 = i;
```

```
            else if (t == "-") { if (!isOp(lt)) i1 = i; }
            else if (t == "*" || t == "/") i2 = i;
            else if (t == "^") { if (i3 == -1) i3 = i; }
            lt = t;
        }

        if (i0 >= 0) {
            p->op = ts[i0];
            p->left = buildExpTree(ts, l, i0 - 1);
            p->right = buildExpTree(ts, i0 + 1, r);
        }
        else if (i1 >= 0) {
            p->op = ts[i1];
            p->left = buildExpTree(ts, l, i1 - 1);
            p->right = buildExpTree(ts, i1 + 1, r);
        }
        else if (i2 >= 0) {
            p->op = ts[i2];
            p->left = buildExpTree(ts, l, i2 - 1);
            p->right = buildExpTree(ts, i2 + 1, r);
        }
        else if (i3 >= 0) {
            p->op = ts[i3];
            p->left = buildExpTree(ts, l, i3 - 1);
            p->right = buildExpTree(ts, i3 + 1, r);
        }
        else if (ts[l] == "-"){
            p = buildExpTree(ts, l + 1, r);
            p->neg = true;
        }
        else {
            assert(ts[l] == "{" || ts[l] == "(");
            p->inGroup = (ts[l] == "(");
            p->left = buildExpTree(ts, l + 1, r - 1);
        }
    }
    return p;
}

Box* getBox(Node* p){
    assert(p);
    const string& o = p->op;
```

```
        Box* b = boxPool.createNew();
        if (o.empty()) { //在括号里面
            assert(p->left); assert(!(p->right));
            b = getBox(p->left), b->addGroup(p->inGroup, p->neg);
        }
        else if (!isOp(o)) {
            b->setGrid(p->neg, o);
        }
        else {
            Box *lb = getBox(p->left), *rb = getBox(p->right);
            if (o == "/") {
                b->setGrid(max(lb->w, rb->w), lb->h + rb->h + 1);
                b->LVC = lb->h;
                int ls = b->w - lb->w; ls = ls / 2 + ls % 2; lb->copyTo(*b, 0,
ls);                    //分子
                int rs = b->w - rb->w; rs = rs / 2 + rs % 2; rb->copyTo(*b, b->LVC
+ 1, rs);               //分母
                b->grid[b->LVC].assign(b->w, '-');
            }
            else if (o == "^") {
                b->setGrid(lb->w + rb->w, lb->h + rb->h); b->LVC = lb->LVC +
rb->h;
                lb->copyTo(*b, rb->h, 0); rb->copyTo(*b, 0, lb->w);
            }
            else {
                assert(ops.count(o));
                int lvc = max(lb->LVC, rb->LVC), h = lvc + max(lb->h - lb->LVC,
rb->h - rb->LVC);
                b->setGrid(lb->w + rb->w + 3, h), b->LVC = lvc;
                lb->copyTo(*b, lvc - lb->LVC, 0), rb->copyTo(*b, lvc - rb->LVC,
lb->w + 3);
                b->grid[b->LVC][lb->w + 1] = o[0];
            }
        }

        return b;
    }

    int main() {
        string line;
        SVec tokens;
       bool first = true;
```

```
    while (cin>>line) {
       if(first) first = false; else cout<<endl;
        tokenize(line, tokens);
        Node* root = buildExpTree(tokens, 0, tokens.size() - 1);
        Box* bp = getBox(root);
        cout<<bp;
        nodesPool.dispose(), boxPool.dispose();
    }
    return 0;
}
```

货运（Trucking, World Finals 1996 – Philadelphia, UVa252，难度 7）

有一个货运网络，其中有很多中转处理中心（下文简称 ICPC）。每个 ICPC 都有一些卸货门，到此地的卡车就在其中一个卸货门卸货。每个 ICPC 也有一些转发门，在此中转的货物就从转发门出发到达下一个 ICPC。

卸货门的数量是有限的，如果不够用，到达的货车就要排队。一个货车可能载有要发往多个 ICPC 的货物。这个队列的排序规则依次如下：

（1）有转发货物的货车比无转发货物的列车更靠前。

（2）同样有转发货物的，转发目的地更远的更靠前。

（3）先到的货车靠前。

不论货物的体积和数量如何，从卸货到转运门的时间就是 2 个小时，在转运门装车的时间不计。只要转运货车装满或当天需要发向转运门对应的 ICPC 的货物已经装完，转运货车就立刻发车。货物数量用货车容量的百分比来衡量，并且为了装满货车，可以被切分成多份。一个转运货车出发和下一辆货车的就绪可以认为一瞬间完成，并且转运货车的数量也是无限的。

输入包含以下数据：

（1）每个 ICPC 的卸货门数量、转发门数量、目标 ICPC 编号、当天需要发出的货物数量以及到达目标 ICPC 的最晚时间。

（2）当天到达的每一个卡车，到达的时间，到达的 ICPC 编号，以及携带的每一个货物。

（3）每一个货物给出其体积、来源以及目标的 ICPC 编号，以及从中转 ICPC 到目标 ICPC 的时间。

现在需要评估：

（1）对于每个 ICPC，在这个卸货的货车的平均等待时间。

（2）哪些货物在运输过程中会迟到。

具体的输入输出格式请参考原题描述。

【分析】

首先，需要用一个优先级队列来管理所有的时间点上的事件。对于每一个 ICPC，也要

建立一个在此等待卸货的货车的队列，队列的排序规则参见题目描述。

全局事件队列中都是一系列的事件，按照时间然后是事件的类型进行排序。事件可以分为 3 种类型：

- ❑ tlArrived //货车到达。
- ❑ rdFree //卸货门可用。
- ❑ smStripped //卸货完成，货物已经到达转运门。

对于 ICPC、中转门以及货物，都建立对应的结构，比较关键的是 ICPC 以及中转门对应的结构：

```
struct ICPC{
    int c, s, d;
    bool stripUsed[maxd];                            //卸货门是否占用
    vector<int> waitingTime;                         //等待时间统计
    vector<Shipment> lateSMs;                        //迟到货物统计
    RelayDoor RDs[maxd];                             //转运门结构
    priority_queue<Trailer> trailerQ;                //到达货车队列
    void strip(const Trailer& t, int time, int sDoor); //对指定的货车卸货
    void stripAll(int time) ;                        //将队列中的所有货车卸货
    void relayAll(int time)                          //所有的转运门发货
    RelayDoor& findRelayDoor(const Shipment& s)      //找到指定货物对应的转运门
}

struct RelayDoor {
    int r, vol, l;
    vector<Shipment>SMs;                              //所有的转运货物
    void relay(int time, vector<Shipment>& lateSMs); //将所有可以转运的货物发出
}
```

然后对 3 种事件的后续操作进行模拟：

- ❑ tlArrived //货车到达，则加入对应 ICPC 的卸货门队列。
- ❑ rdFree //卸货门可用，设置对应卸货门的状态。
- ❑ smStripped //卸货完成，货物已经到达转运门。

☞注意：

载有转运货物的货车的转运距离，依据其所载货物的转运时间来判断。

窗口管理器（Window Manager, World Finals 2015 – Marrakech, LA7162，难度 7）

现在要设计一种手机屏幕上使用的窗口管理器。屏幕是高宽为 x_{max} 和 y_{max} 的矩形（$0<x_{max}, y_{max}\leqslant 10^9$），左上角的坐标是(0,0)。上面显示了一些窗口。窗口的边界不能超出屏幕边缘，也不能互相覆盖。管理器支持如下命令（其中，$0\leqslant x<x_{max}$，$0\leqslant y<y_{max}$，$1\leqslant w, h\leqslant 10^9$，$|d_x|,|d_y|\leqslant 10^9$）。

- OPEN x y w h：创建一个窗口，左上角坐标是(x,y)，宽 w 像素，高 h 像素。
- CLOSE x y：关闭包含像素(x,y)的窗口。
- RESIZE x y w h：设置包含像素(x,y)的窗口的高宽为 w 和 h。窗口的左上角坐标不动。
- MOVE x y d_x d_y：移动包含像素(x,y)的窗口。距离是水平方向的 d_x 或者垂直方向为 d_y，d_x 和 d_y 最多会有一个不为 0。

只有结果窗口不覆盖其他窗口或者超越屏幕边缘，OPEN 和 RESIZE 命令才会成功。MOVE 命令会尽量使得窗口移动的距离接近命令指定的像素数。例如，如果 d_x 是 30，但是窗口只能向右移动 15 像素，那么就移动 15 像素。

一个窗口可能会撞到其他窗口，图 3.11 中窗口 A 会把 B 往要移动的方向尽量推。这个行为还会传递，B 还会推 C，以此类推。

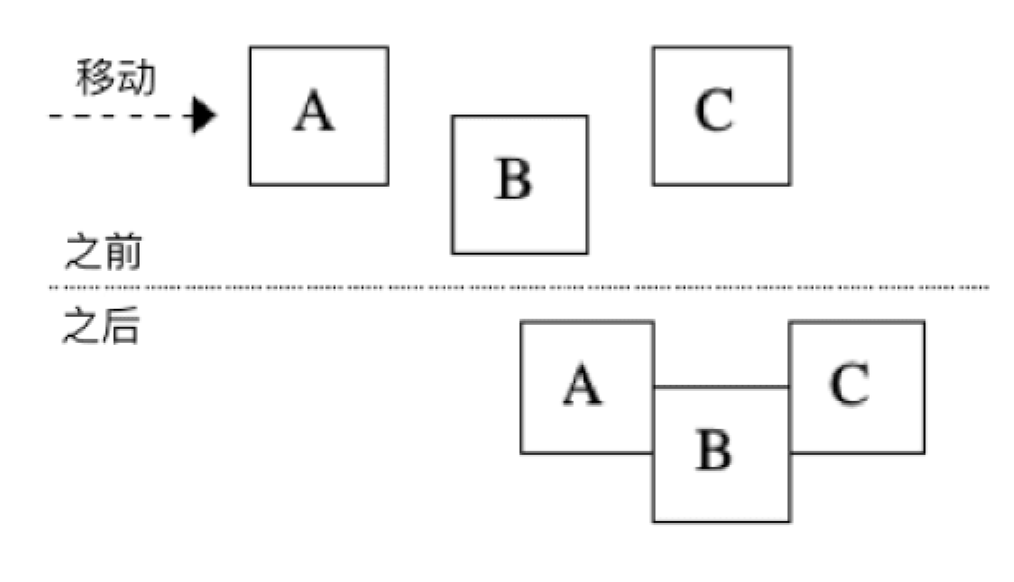


图 3.11

输入不超过 256 条上述命令，依次执行，按照样例中的格式输出执行结果。如果执行命令遇到错误，输出命令的编号、命令名称以及下面错误信息中的第一个合适的，并且忽略这条命令。

- no window at given position – 对于 CLOSE、RESIZE 和 MOVE 命令：如果不存在包含指定位置像素的窗口。
- window does not fit - 对于 OPEN 和 RESIZE 命令：如果结果窗口会覆盖其他窗口或者超出屏幕边缘。
- moved d’ instead of d - 对于 MOVE 命令：如果命令要求移动 d 个像素，但是为了不超出屏幕边缘，只能移动 d'像素。这种情况下，屏幕仍然移动，只是距离短一些。

所有命令执行完成，错误信息输出之后，给出仍然打开的窗口的数量。接着对于每个窗口，根据创建的顺序，输出窗口的左上角坐标以及高度和宽度。

样例输入：

```
320 200
OPEN 50 50 10 10
OPEN 70 55 10 10
OPEN 90 50 10 10
RESIZE 55 55 40 40
RESIZE 55 55 15 15
MOVE 55 55 40 0
CLOSE 55 55
```



```
CLOSE 110 60
MOVE 95 55 0 -100
```

样例输出：

```
Command 4: RESIZE - window does not fit
Command 7: CLOSE - no window at given position
Command 9: MOVE - moved 50 instead of 100
2 window(s):
90 0 15 15
115 50 10 10
```

【分析】

窗口数目不超过 256，所以维护一个当前打开的窗口列表，每次需要查找一个窗口或者判断新建的窗口是否与现有的窗口重叠时，进行线性遍历即可。

这里可能造成困难的是判断窗口重叠的逻辑：对于窗口 $W1(x1,y1,w1,h1)$ 和 $W2(x2,y2,w2,h2)$ 来说，二者有公共点的充要条件就是区间 $[x1, x1+w1-1]$ 和 $[x2,x2+w2-1]$ 有公共点且 $[y1,y1+h1-1]$ 和 $[y2,y2+h2-1]$ 有公共点。读者可以参考图 3.12 思考一下。

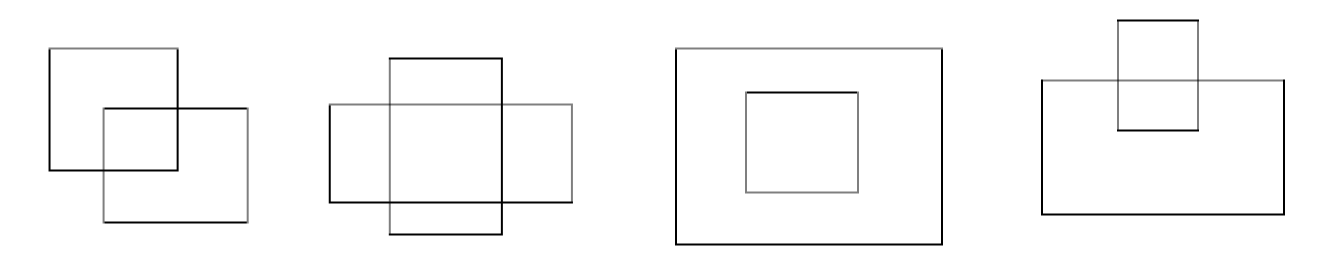


图 3.12

比较麻烦的是 MOVE 操作。首先考虑向右移动，可以先将窗口按照左上角的 x 坐标进行递增排序，然后从右到左依次遍历每个窗口看能向右移动的最大距离是多少。对于每个窗口，遍历它向右移动时可能碰到的所有窗口（右边界也要考虑），即可得到向右移动的最大值。然后执行实际的移动操作，移动的距离不能超过之前计算出来的最大值。一个窗口移动时要递归把可能要碰到的窗口移动好，然后再移动自身。具体的实现过程请参见代码。

当右移逻辑完成之后，对于其他逻辑可以通过把整个图形和所有窗口向右旋转 90°、180° 或 270° 之后，调用右移逻辑，然后再向右旋转回原来的方向即可。

需要注意的是，本题中的矩形不是几何上的矩形，实际可以认为是一些边长为 1 的正方形（像素点）的集合。

完整程序（C++11）如下：

```
using namespace std;
bool inRange(int x, int l, int r) { return  x >= l && x < r; }
bool intersect(int l1, int r1, int l2, int r2) { //[l1, r1) 和 [l2, r2)两
个区间是否有公共点
    assert(l1 < r1 && l2 < r2);
    return inRange(l1, l2, r2) || inRange(l2, l1, r1) || inRange(r1 - 1, l2,
r2) || inRange(r2 - 1, l1, r1);
```

```
}

int XMAX, YMAX, cmdIdx, winIdx;
inline bool outOfBound(int px, int py) {
   return !inRange(px, 0, XMAX) || !inRange(py, 0, YMAX);
}
struct Window;
typedef vector<Window> VW;
typedef VW::iterator VWI;
struct Window {
   int x, y, w, h, idx, maxMove;
   Window(int _x, int _y, int _w, int _h) : x(_x), y(_y), w(_w), h(_h) {}
   bool containsPt(int px, int py) const {
      return inRange(px, x, x + w) && inRange(py, y, y + h);
   }
   bool overlap(const Window& w2) const {
      return intersect(x, x + w, w2.x, w2.x + w2.w) && intersect(y, y + h,
w2.y, w2.y + w2.h);
   }
   bool outOfBound() const {
      return ::outOfBound(x, y) || ::outOfBound(x, y + h - 1)
      || ::outOfBound(x + w - 1, y) || ::outOfBound(x + w - 1, y + h - 1);
   }
   bool VCross(int yl, int yr) { return intersect(yl, yr, y, y + h); }
//[yl, yr)这个垂直区间和w所在的垂直区间有公共点吗
   vector<VWI> crossWins;
};

struct Command{
   string cmdText;
   int x, y, dx, dy, idx;
   Command(const string str, int _x, int _y, int _dx, int _dy) :
      cmdText(str), x(_x), y(_y), dx(_dx), dy(_dy), idx(cmdIdx++) {}
};

vector<Command> cmds;
VW wins;
void rotate90(int& px, int& py) { //将整个图形旋转90°之后，px和py的新坐标是什么
   int ny = px, nx = YMAX - py - 1; px = nx, py = ny;
}
void rotate90() { //把整个图形旋转90°
   for (auto& w : wins) {
```

```
            int nx = w.x, ny = w.y+w.h-1;
            rotate90(nx, ny);
            swap(w.w, w.h); w.x = nx, w.y = ny;
        }
        swap(XMAX, YMAX);
    }

    void init() { cmdIdx = 1; winIdx = 0; cmds.clear(); wins.clear(); }

    VWI findWindow(int px, int py) {
        for(auto p = wins.begin(); p != wins.end(); p++)
            if (p->containsPt(px, py)) return p;
        return wins.end();
    }

    VWI findWindow(const Command& cmd, ostream& os){
        VWI pw = findWindow(cmd.x, cmd.y);
        if (pw == wins.end())
            os<<"Command "<<cmd.idx<<": "
                <<cmd.cmdText<<" - no window at given position"<<endl;
        return pw;
    }

    bool windowFit(const Window& nw, int existWinIdx = -1) {
        return !nw.outOfBound()
            && all_of(wins.begin(), wins.end(), [nw, existWinIdx](const Window & w)
                { return w.idx == existWinIdx || !w.overlap(nw); });
    }

    void openWindow(const Command& cmd, ostream& errOs) {
        Window nw(cmd.x, cmd.y, cmd.dx, cmd.dy);
        if (!windowFit(nw)) {
            errOs<<"Command "<<cmd.idx<<": "<<cmd.cmdText<<" - window does not
fit"<<endl;
            return;
        }
        nw.idx = winIdx++;
        wins.push_back(nw);
    }
    void resizeWindow(const Command& cmd, ostream& errOs) {
        VWI pw = findWindow(cmd, errOs);
        if (pw == wins.end()) return;
```

```
        Window nw(pw->x, pw->y, cmd.dx, cmd.dy);
        if (!windowFit(nw, pw->idx)) {
            errOs<<"Command "<<cmd.idx<<": "<<cmd.cmdText<<" - window does not
fit"<<endl;
            return;
        }
        pw->w = cmd.dx, pw->h = cmd.dy;
    }
    void closeWindow(const Command& cmd, ostream& errOs) {
        VWI pw = findWindow(cmd, errOs);
        if (pw == wins.end()) return;
        wins.erase(pw);
    }
    void moveRight(VWI p, int dist) {
        for(auto p2 : p->crossWins){
            int p2d = p2->x - (p->x + p->w);
            assert(p2d >= 0);
            assert(p->VCross(p2->y, p2->y+p2->h));
            if(p2d < dist) moveRight(p2, dist-p2d);
        }
        p->x += dist;
    }

    void moveRight(const Command& cmd, ostream& os) {
        for(auto& w : wins) w.maxMove = 0, w.crossWins.clear();
        sort(wins.begin(), wins.end(), [](const Window & w1, const Window & w2)
            { return w1.x < w2.x || (w1.x == w2.x && w1.y < w2.y); });
        VWI pw = findWindow(cmd.x, cmd.y);
        assert(pw != wins.end());
        assert(cmd.dx > 0 && cmd.dy == 0);
        for (auto p = wins.end() - 1; p >= pw; p--){
            int move = XMAX - (p->x + p->w);
            for (auto p2 = p + 1; p2 < wins.end(); p2++) {
                int p2d = p2->x - (p->x + p->w);
                if (p2d >= 0 && p->VCross(p2->y, p2->y + p2->h)){
                    move = min(move, p2d + p2->maxMove);
                    p->crossWins.push_back(p2);
                }
            }
            p->maxMove = move;
        }
        int d = min(pw->maxMove, cmd.dx);
```



```
    moveRight(pw, d);
    if (d != cmd.dx)
        os<<"Command "<<cmd.idx<<": "<<cmd.cmdText<<" - moved "
            <<d<<" instead of "<<cmd.dx<<endl;
}

void moveWindow(const Command& cmd, ostream& os) {
    if (findWindow(cmd, os) == wins.end()) return;
    assert(cmd.dx == 0 || cmd.dy == 0);
    Command ncmd = cmd;
    int r = 0;
    if (cmd.dy == 0) {
        assert(cmd.dx);
        if (cmd.dx < 0)                                  //向左移动
            ncmd.dx = -cmd.dx, r = 2;                    //右转 180°
    } else {
        if (cmd.dy > 0)                                  //向下移动
            ncmd.dx = cmd.dy, ncmd.dy = 0, r = 3;        //右转 270°
        else //向上移动
            ncmd.dx = -cmd.dy, ncmd.dy = 0, r = 1;       //右转 90°
    }

    _for(i, 0, r) rotate90(ncmd.x, ncmd.y), rotate90();
    moveRight(ncmd, os);
    _for(i, 0, (4 - r) % 4) rotate90();
}

void solve() {
    for (const auto& c : cmds) {
        if (c.cmdText == "OPEN") openWindow(c, cout);
        else if (c.cmdText == "CLOSE") closeWindow(c, cout);
        else if (c.cmdText == "RESIZE") resizeWindow(c, cout);
        else if (c.cmdText == "MOVE") moveWindow(c, cout);
    }
    cout << wins.size() << " window(s):" << endl;
    sort(wins.begin(), wins.end(),
        [](const Window & w1, const Window & w2){ return w1.idx < w2.idx; });
    for(const auto& w : wins) cout<<w.x<<" "<<w.y<<" "<<w.w<<" "<<w.h<<endl;
}

int main() {
    string line, buf;
    bool first = true;
    while (true) {
        if (!getline(cin, line)) { solve(); break; }
```

```
        stringstream ss(line);
        if (isdigit(line[0])) {
            if (first) first = false; else solve();
            ss>>XMAX>>YMAX;
            init();
        }
        else {
            int x, y, w, h;
            ss>>buf>>x>>y; if (buf != "CLOSE") ss>>w>>h;
            cmds.push_back(Command(buf, x, y, w, h));
        }
    }
    return 0;
}
```

ASCII 表达式（ASCII Expression, ACM/ICPC Asia - Fukuoka 2011, LA5858，难度 8）

旧式的论文中，数学公式打印出来占多行，所有字符都用等宽字体打印。例如，$\left(1-\frac{4}{3^2}\right)^2 \times -5+6$ 会被打印成如下 4 行：

```
        4   2
( 1 - ---- ) * - 5 + 6
        2
       3
```

其中-5 表示 5 前面加一个一元的负数运算符，这种表达式称为“ASCII 表达式”。其结构的构造规则和 BNF 类似，详情如下，注意语法上需要的特殊空格用 . 表示。

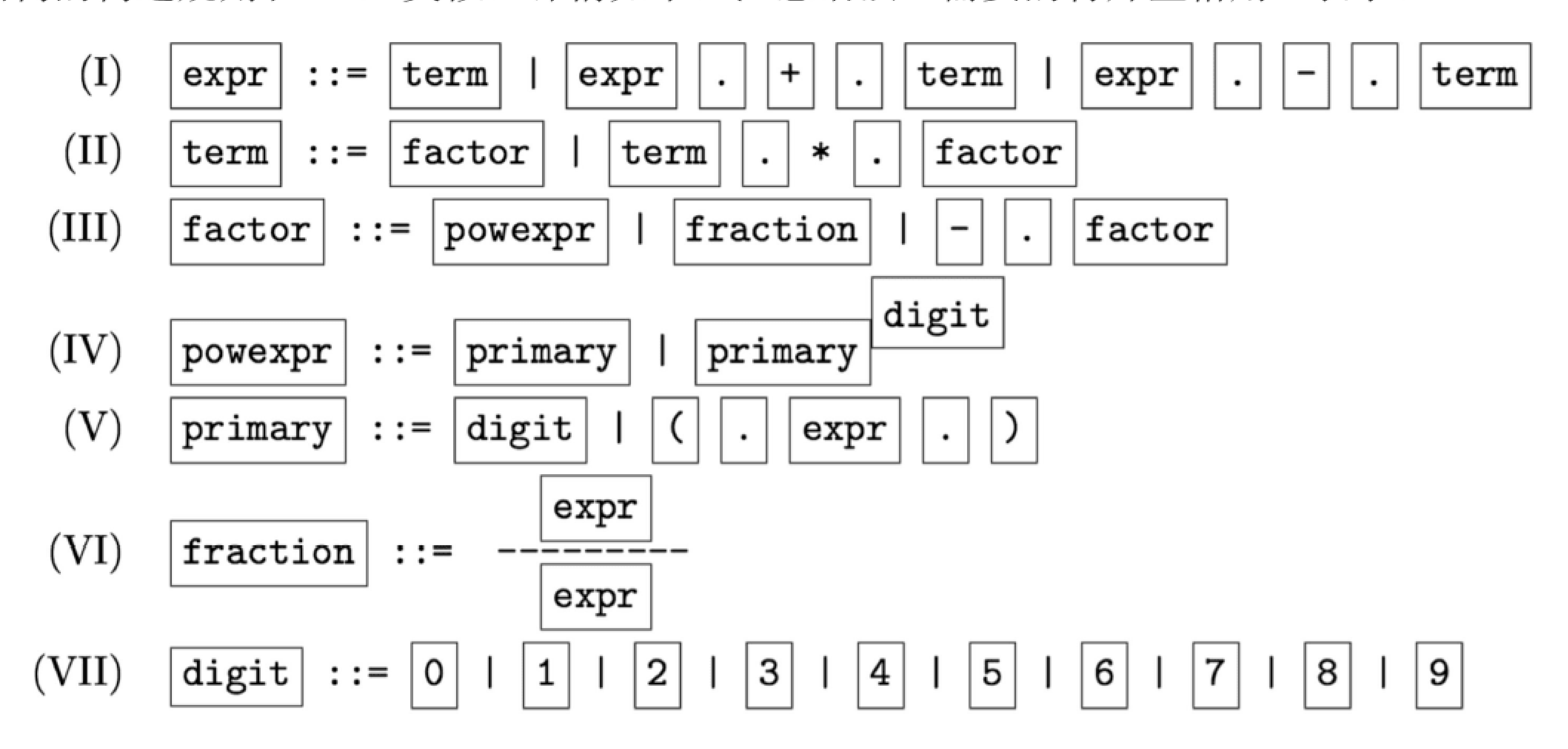


图 3.13 给出了几种符号的顶线（top）、基线（base）和底线（bottom）。分别是表达



式$1-\frac{4}{3^2}$、分式$\frac{4}{3^2}$、幂式 3^2 和数字 1。

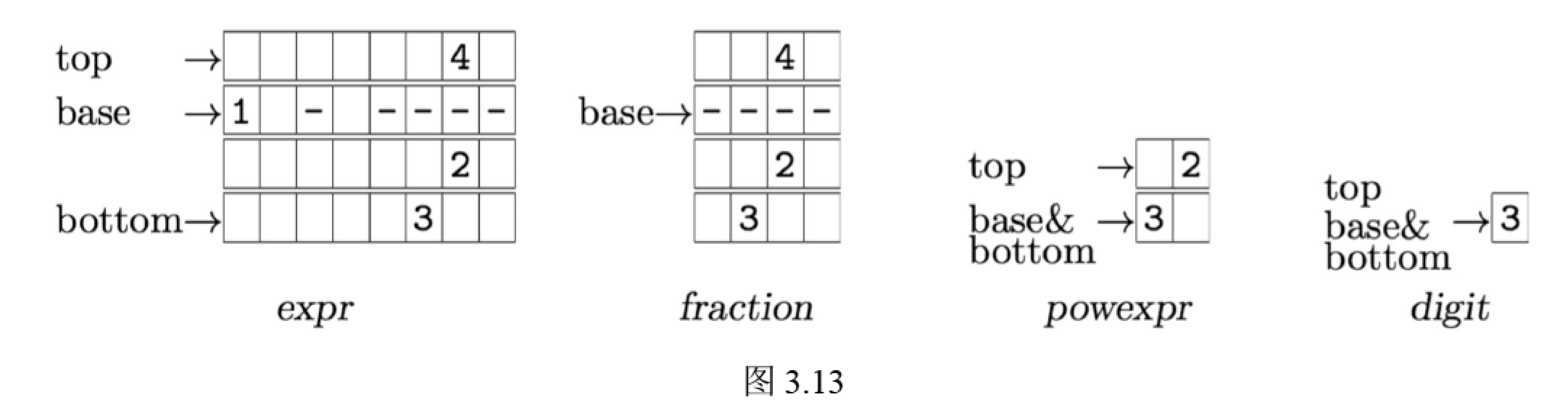


图 3.13

（1）终端符号有'0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'、'+'、'−'、'*'、'(', ')'以及' '。

（2）非终端符号有 expr（表达式）、term（项）、factor（因子）、powexpr（幂）、primary（主表达式）、fraction（分式）以及 digit（数字）。最初的符号就是 expr。

（3）一个格子就是包含字符的一个矩形区域，它们对应于一个字符或者符号。对应于一个终端符号的格子只包含一个单字符。对应于非终端符号的格子包含其他格子作为其后代，但是格子之间不会互相重叠。

（4）每个格子都有一个基线（base line）、顶线（top line）和底线（bottom line）。I、II、III 和 V 规则的右边格子的基线应该和外层格子对齐。它们的垂直位置决定了左边格子的基线。

（5）幂表达式包含底数和一个可选的数字。数字放在底数格子的上面一行，水平相邻。幂表达式的基线和底数相同。

（6）分式的分数线包含 3 个以上的连续横线字符，横线上是除数的表达式，横线下是被除数。横线的长度 wh，等于 $2+\max(w1,\ w2)$，其中 $w1,w2$ 分别表示分子和分数格子的长度。二者都是居中，左边有$\lceil (wh-wk)/2\rceil$个空格，右边有$\lfloor (wh-wk)/2\rfloor$个空格，其中（k=1,2）。分式的基线就是分数线。

（7）数字仅包含 1 个字符。

举例来说，负分数$-\frac{3}{4}$用三行表示：

```
   3
 - ---
   4
```

作为一元运算符的负号和分数线之间要有一个空格。

分式$-\frac{3+4\times -2}{-1-2^2}$用四行字符表示：

```
 3 + 4 * - 2
-------------
          2
   - 1 - 2
```

分子分母格子的宽度是 11 和 8，所以分数线的长度是 2+max(11,8)=13。分母靠左$\lceil(13-8)/2\rceil=3$个空格居中，靠右$\lfloor(13-8)/2\rfloor=2$个空格。

幂表达式$(4^2)^3$用 2 行字符表示：

```
  2 3
( 4 )
```

其中 2 的格子被放到 4 的格子的基线上方，3 的格子放到底数（4^2）的上方。

现在需要编程识别 ASCII 表达式的结构并求值。注意本题中除法定义为模 2011 的逆。

【分析】

关键是对于题目中给出的形如 BNF 的规则的理解，BNF 实际上是一种递归形式的规则定义：如 expr 实际上就是很多的 term 用+/−连接起来的表达式。同理 term 就是很多 factor 用*号连接起来的乘积。

观察任一区域，首先要找到基线：从左到右，依次从上到下遍历到的第一个非“.”字符所在的行就是。然后在递归计算每一个区域时就可以根据基线上面的首字符来区分当前的区域是何种表达式并且做相应的计算。尤其，在计算 fraction 时，实际上分数线上下就是两个独立的区域，分别按照 expr 进行解析计算即可。

另外因为题目里面不牵涉字母，全是一位的数字，所以可以直接递归计算而不再需要建立表达式树。可事先将小于 2011 的每一个数模 2011 的逆遍历保存下来。

完整程序如下：

```
using namespace std;

const int MOD = 2011;
int INV[MOD + 5], N, W;
string S[24];

int base_line(int top, int bottom, int left, int right);
int eval_fraction(int base, int& pos, int top, int bottom);
int eval_primary(int base, int& pos, int top, int bottom);
int eval_powexpr(int base, int& pos, int top, int bottom);
int eval_factor(int base, int& pos, int top, int bottom);
int eval_term(int base, int& pos, int top, int bottom);
int eval_expr(int base, int& pos, int top, int bottom);
int eval(int, int, int, int);

void dbgPrint(int base, int pos, int top, int bottom, const char* log) {
   return;
   char buf[128];
   sprintf(buf, " , %d-, (%d), (%d, %d)", base, pos, top, bottom);

   cout<<log<<buf<<endl;
```

```
        _for(i, top, bottom){
            if(i == base) cout<<"->"; else cout<<"  ";
            for(int j = pos; j < S[i].size(); j++) {
                char c = S[i][j]; if(c == '.') c = ' ';
                cout<<c;
            }

            cout<<"|"<<endl;
        }
    }

    void dbgPrint(int top, int bottom, int left, int right) {
        return;
        //cout<<"Region: "<<endl;
        _for(j, top, bottom){
            _for(i, left, right) cout<<S[j][i];
            cout<<endl;
        }
    }

    //找到第一个非'.'字符所在的基线
    int base_line(int top, int bottom, int left, int right) {
        assert(right <= W);
        _for(i, left, right) _for(j, top, bottom) if(S[j][i] != '.') return j;
        assert(false);
    }
    //分式求值
    int eval_fraction(int base, int& pos, int top, int bottom) {
        int left = pos+1;
        //dbgPrint(base, pos, top, bottom, "eval_fraction");
        assert(S[base][pos] == '-');
        while(S[base][pos] == '-') pos++;
        pos++;
        return eval(top, base, left, pos-2) * INV[eval(base+1, bottom, left,
pos-2)] % MOD;
    }
    //主表达式求值
    int eval_primary(int base, int& pos, int top, int bottom) {
        //dbgPrint(base, pos, top, bottom, "eval_primary");
        char cp = S[base][pos];
        if(isdigit(cp)) { pos += 2; return cp - '0'; }
        assert(cp == '(');
```

```
        pos += 2;
        int ret = eval_expr(base, pos, top, bottom);
        assert(S[base][pos] == ')');
        pos += 2;
        return ret;
    }

    //求幂
    int eval_powexpr(int base, int& pos, int top, int bottom) {
        //dbgPrint(base, pos, top, bottom, "eval_primary");
        int ex = eval_primary(base, pos, top, bottom);

        if(pos - 1 < W && S[base][pos-1] == '.' && base > top && isdigit
(S[base-1][pos-1])) {
            int p = S[base-1][pos-1] - '0', ret = 1;
            _for(i, 0, p) ret = (ret * ex) % MOD;
            pos++;
            return ret;
        }

        return ex;
    }
    //因子求值
    int eval_factor(int base, int& pos, int top, int bottom) {
        //dbgPrint(base, pos, top, bottom, "eval_factor");
        if(S[base][pos] == '-' && S[base][pos+1] == '-'){  //分式
            return eval_fraction(base,pos,top,bottom);
        } else if(S[base][pos]=='-') {  //负因子
            assert(S[base][pos+1] == '.');
            pos += 2;
            return (MOD - eval_factor(base,pos,top,bottom)) % MOD;
        } else {  //幂
            return eval_powexpr(base,pos,top,bottom);
        }
    }

    int eval_term(int base, int& pos, int top, int bottom) {
        assert(pos < W);
        //dbgPrint(base, pos, top, bottom, "eval_term");
        int ret = eval_factor(base, pos, top, bottom);
        while(pos < W && S[base][pos] == '*'){
            pos += 2;
```



```
        ret = (ret * eval_factor(base, pos, top, bottom)) % MOD;
    }

    return ret;
}

int eval_expr(int base, int& pos, int top, int bottom) {
    //dbgPrint(base, pos, top, bottom, "eval_expr");
    int ret = eval_term(base, pos, top, bottom);
    while(pos<W) {
        if(S[base][pos] == '+') {
            pos += 2;
            int term = eval_term(base, pos, top, bottom);
            ret = (ret + term) % MOD;
        }
        else if(S[base][pos] == '-') {
            pos += 2;
            int term = eval_term(base, pos, top, bottom);
            ret = (ret - term + MOD) % MOD;
        } else {
            break;
        }
    }

    return ret;
}

int eval(int top, int bottom, int left, int right) {
    int base = base_line(top, bottom, left, right), i;
    for(i = left; i < right; i++) if(S[base][i] != '.') break;
    //dbgPrint(base, i, top, bottom, "eval");
    return eval_expr(base, i, top, bottom);
}

int main(){
    //先把每个数关于模 2011 的逆求出来
    _rep(i, 1, MOD) _rep(j, 1, MOD) if((i*j)%MOD == 1) INV[i] = j;
    while(cin>>N && N){
        _for(i, 0, N) cin>>S[i], assert(S[i].length() == S[0].length());
        W = S[0].length();
        cout<<eval(0, N, 0, W)<<endl;
    }
```

```
    return 0;
}
```

拖拉机游戏模拟（Game Simulator, Asia - Shanghai 2009, LA4749，难度 9）

拖拉机是在中国非常流行的一种扑克游戏。有 4 个玩家，按照顺时针顺序的名字依次是 Alice、Bob、Charles、David，下文简称 A、B、C、D。玩家分成两队，A 和 C 是 1 队的，另两个是 2 队的。游戏中要用到两副共 108 张牌。本题中使用一种简化的游戏规则。

游戏是回合制。每一回合，一队称为庄家方（CT），另一队称为抓分方（FT）。每一队都有一个当前级别（CR, A,2,3…J,Q,K 等）。玩家的目的是让自己所在的队尽快升级。每回合都有一个主要的花色（红心❤-H，黑桃♠-S，梅花♣-C，方块♦-D，或者无花色-O）和主级 CR。这一轮的主牌就是 CT 的 CR 对应的牌，并且花色由 CT 给出。花色和主级会决定这一轮每张牌的大小顺序。

牌 5、T(10)、K 分别对应 5、10、10 分，其他的牌都是 0 分。每轮中只有 FT 需要得分，规则在下文中讨论。如果在一轮中 FT 得分小于 80，下一轮必须继续当 FT。这种情况称为“保级”。否则，他们在下一轮变成 CT，并且原来的 CT 变成 FT，这种情况称为“下台”。

FT 的得分以及对应的双方升级的规则如下：

（1）得 0 分，CT 升 3 级。例如，如果 CT 当前是 9 级，就升到 Q(12)。

（2）小于 40，CT 升 2 级。

（3）小于 80，CT 升 1 级。

（4）大于等于 $80+k*40$ 并且小于 $120+k*40$，FT 就升 k 级。例如 FT 得 255 分，FT 升 4 级。

（5）80 分，两队都不升级。

如果某个队的级别升到大于 A（A 比 K 高一级），这个队就是整个游戏的赢家。

在一轮中，CT 中有一个玩家称为庄家。如果“保级”，庄家的队友变成下一轮的庄家。如果下台，庄家右手边的下家成为下一轮的庄家。例如，这一轮的庄家是 A 且庄家方下台，那么下一轮的庄家就是 A 右手边的下家 B。

回合一开始，庄家之外的每个玩家拿到 25 张牌，庄家拿到剩下的 33 张牌。之后庄家选择 8 张交给裁判，这些牌称为底牌。

现在每个玩家刚好有 25 张牌。一回合包含多圈。第一圈中，庄家打出一到多张牌（首牌），接着按照顺时针顺序，其他人必须依次打出跟庄家同样数量的牌（称为“跟牌”）。当前这圈的胜者在下一圈先出牌，以此类推。如果有一圈的胜者是 FT 的成员，那么这一圈中打出来所有牌的分数之和作为 FT 的得分。

接下来描述如何确定每一圈的胜者：

在一轮的花色和 CR 确定下来之后，就可以确定主牌，也就是说符合当前花色或者牌面等于 CR 的所有主牌以及大小王。所有其他的牌都是副牌。

所有的牌按照如下规则排序：



（1）主牌比副牌大。

（2）对于主牌，顺序如下：大王 RJ >小王 BJ >符合花色的主级牌（如果存在）>其他主级牌>其他的牌按照牌面排序（A, K, Q, J, T, 9, 8, 7, … , 3, 2）。

（3）对于副牌，就按照牌面排序。

假设在下文的描述中，这一轮的 CT 的 CR 是 7 级。主花色是 H，那么所有的牌如下排序：

```
S2 , C2 , D2
< S3 , C3 , D3
< S4 , C4 , D4
< S5 , C5 , D5
< S6 , C6 , D6
< S8 , C8 , D8
< S9 , C9 , D9
< ST , FT , CT (T - 10)
< SJ , CJ , DJ (J - Jack)
< SQ , CQ , DQ (Q - Queen)
< SK , CK , DK (K - King)
< SA , CA , DA (A - Ace)
< H2
< H3
< H4
< H5
< H6
< H8
< H9
< HT
< HJ
< HQ
< HK
< HA
< S7 = C7 = D7
< H7
< BJ (the Black Joker)
< RJ (the Red Joker)
```

如果这一轮没有主花色，那么牌的顺序如下：

```
H2 , S2 , C2 , D2
< H3 , S3 , C3 , D3
< H4 , S4 , C4 , D4
< H5 , S5 , C5 , D5
< H6 , S6 , C6 , D6
```

< H8 , S8 , C8 , D8
< H9 , S9 , C9 , D9
< HT , ST , FT , CT
< HJ , SJ , CJ , DJ
< HQ , SQ , CQ , DQ
< HK , SK , CK , DK
< HA , SA , CA , DA
< H7 = S7 = C7 = D7
< BJ(大王)
< RJ(小王)

上面的牌中，斜体的是主牌，粗体的是副牌。

每一圈，第一个人打出来的首牌，必须要么都是主牌，或者都是同色的副牌。所有可能的牌的结构如下（假设主花色是 H 并且当前例子中主级是 7）。

（1）单张牌：如 D9。

（2）对牌：两张完全相同的牌。D9D9 是，D7S7 不是。

（3）拖拉机：两个或者以上按照上文所述顺序是连续的对牌，并且要求都是主牌或者都是同色的副牌，如 SJSJSQSQSKSKSASA、H7H7S7S7HAHA、RJRJBJBJ。以下都不是拖拉机：S7S7C7C7（大小不是连续的），C7C7C6C6（主牌和副牌），DADAD2D2（A 不是 1，所以不是连续的），H2H2H4H4，D2D2D3。如果花色是无，那么 H7H7S7S7HAHA 就不是拖拉机（因为无花色，所以 H7 和 S7 大小相同，不是连续的）。

（4）甩牌：上述所有结构的组合，如果都是主牌或者同色的副牌，就可以放在一起甩。在本题中，单牌、对牌、拖拉机的任何组合都可以扔出来。例如，RJRJBJBJH7H7HQHQHJHJH9H9H6H6HAH2 包含 6 个部分：两个拖拉机，两对和两张单牌（RJRJBJBJH7H7-HQHQHJHJ-H9H9-H6H6-HA-H2）；CACAC8C8CK 包含 3 部分：两对加一张单牌（CACA-C8C8-CK）。

甩牌可以认为是不同部分的组合：H2H2H3H3H4H4H5H5H6H6 可以认为是 H2H2H3H3-H4H4H5H5H6H6 或者 H2H2-H3H3H4H4H4H5H5-H6H6 等。对于每一圈的首牌，拆分时每次都选择其中最长的子结构（同样长选择最大的）来构造组合，就构成首牌的结构，也就是这一圈的结构。所以每一圈的牌的结构是唯一的。

当第一个玩家出牌之后，其他玩家就按照顺时针顺序依次出牌。

游戏有个重要部分就是确定每一圈的赢家：

（1）如果某个玩家的牌同时包含“主牌”和“副牌”或者是不同花色的副牌，他就不能是胜者。

（2）如果首牌都是副牌并且之后某个人的跟牌包含同首牌不同颜色的副牌，这个人不是胜者。

（3）如果某人的跟牌无法构造成和首牌一样的结构，这个玩家也不能是胜者。

（4）否则，如果这一圈牌的结构不是甩牌，打出最大牌的玩家赢得这一圈。如果多个玩家都打出同样大小的牌，这一圈的胜者就是先出牌的那个。



接下来考虑甩牌的情况。对于跟牌的结构，按照首牌的结构进行构造，尽量使得所有最长结构中的最大牌最大化（这张牌称作荣牌）。注意拖拉机可被视为是多个更短的拖拉机的组合，并且对牌可以认为是两张单牌。一圈的胜者就是打出最大荣牌的那个人。如果有多人，先出者胜。如果首牌甩出来一堆副牌，唯一一种打赢它的规则就是扔出结构相同的主牌，并且不可能打败首牌中甩出的主牌。

关于底牌的规则如下：如果最后一圈的胜者是 FT 的一员，那么 FT 就得到底牌中所有牌的分数乘以 2^w。如果最后一圈不是甩牌，那么 w 就是最后一圈中首牌的张数。如果是甩牌，w 就是首牌结构中最长的结构，例如“RJRJBJBJH7H7HQHQHJHJH6H6HA”，w 就是 6，因为其中的最长结构是“RJRJBJBJH7H7”，长度为 6。

表 3.4 所示的例子中，A 打首牌，并且假设在本例中，当前打 7 级，主花色是 H。

表 3.4

A	B	C	D	胜者	注　　释
SA	S2	ST	S5	A	A 打出最大的 A
SA	S2	ST	SA	A	A 打出第一个黑桃 A
SA	S2	ST	H2	D	D 打出第一个也是唯一的主牌
SA	H2	C7	D7	C	C 打出主♣7，D 打出♦7，跟♣7 大小一样
C2C2	C3C4	C7D7	RJBJ	A	这一圈的结构是对牌，除了 A 其他玩家都是两张单牌
D3D3	DTDT	SKSK	H2H3	B	B 打出一对牌比 A 的更大，同时 C 打出花色不同的对牌
D3D3	DTDT	SKSK	H2H2	D	D 打出唯一的一对主牌
D6D6D8D8	DJDJDKDK	DTDTD2D3	HTHTBJBJ	A	A 打出唯一的拖拉机（当前的主级是 7）
H6H6H8H8	H7H7BJBJ	C2C2C3C4	HKHKRJRJ	B	B 的拖拉机比 A 的更大
H6H6H8H8	H7H7D7D7	C2C2C3C4	HKHKRJRJ	B	B 也打了个拖拉机
SASK	STST	C2H3	S7SK	A	A 做了个甩牌
SASK	HKH3	HAH2	S7SK	C	B 和 C 都比 A 大
SASK	HAH2	HAH3	S7SK	B	B 和 C 都比 A 大，但是 B 的红桃 A 出的更早
S2S2S3S3SA	H3H3H4H4RJ	D7D7H7H7H2	S7S7SQSJS6	C	B 和 C 都比 A 大

编写一个只玩一轮的拖拉机游戏模拟器。

输入描述：

有 T 个测试案例。对于每个案例：

首先给出这一轮的主花色（H,S,C,D,O；“O”表示无花色）；这一轮的庄家名称，1 队的级别，2 队的级别。接下来的每一行给出 4 个字符串，依次是首家和其他玩家按照顺序打出的牌。每个字符串中对应每张牌的顺序不定。每个玩家会刚好打出 25 张牌。可以假设输入一定有效。

输出描述：

输出 FT 一方在这一轮的得分。如果某一队在这一轮之后赢了整个游戏，输出“Winner: Team X”。X 是 1 或 2。否则输出队 1 和队 2 在这一轮之后的新级别，然后输出下一轮的庄家方。详细的输出格式请参见样例输出。

样例输入：

```
1
O Charles 2 2
S6S6S7S7 SASKSJST STS8S4S4 S3S5SJSQ
S9S9 H3D3 S3DT SAD3
DA DQ DK D4
SKS8S5S3 RJC2D2H2 C6C8CJD9 H3CKDTD5
H7H7 H6H4 HJHQ H9H9
DJDJ DKH5 D5D4 D6D6
D8D8 C4C3 HTH5 D9D7
C5C5 C6CT H8HQ C7C4
H8 C7 HA HA
H2 RJ BJ CK
DA BJ C8 HK
S2S2C2 CQCAD2 HTHJHK C9CQCA
```

样例输出：

```
Case #1:
50
3 2 Alice
```

【分析】

首先最重要的是牌的排序，因为规则非常繁杂，所以如果写成大量的逻辑判断会非常的麻烦。在读取每一轮的主级和花色之后，可以根据题目给出的规则依次从大到小给每张牌指定一个序数，在后面比较两张牌的顺序时直接使用序数比较即可。在判断拖拉机时也可以看看相邻的两对牌序数是否相差 1 即可。

接下来在每一圈的出牌之后，首先是把每一组牌按照序数进行升序排序。这样对首牌进行结构拆分时，可以按照题目所述的规则不停从左到右查找最大的结构，第一次查到的结构自然就是最大的。首牌结构拆分完了之后，就是判断后面每一组跟牌的大小，首先排除不能为胜者的牌，对这一组牌进行拆分时，未必每次都选择最大的结构，而是要按照首牌拆出来的结构每一次进行决策然后进行回溯，最后看是否能够拼出和首牌一样的结构，详见后文代码中的 getStructDfs 函数。拼出合适的结构之后再根据 Hornor Card 计算这一局的胜者以及对应 FT 的得分。

另外，在读取所有的输入之后就可以计算出所有底牌及其对应的张数，在最后一轮之后计算 FT 在底牌上的得分。

如下面的测试案例：

```
    H David A A
    SKSKSQSQSTSTS9S9S5S5S6S6S7S7 HKHKHQHQHJHJHTHTH8H8H7H7H6H6 CAC2C2C3C3C4C4C
5C5C 6C6C7C7C8 DAD2D2D3D3D4D4D5D5D6D6D7D7D8
    RJ C8 D8 SA
    HAHAH3H3H4H4H5H5H9H9 C9C9CTCTCJCJCQCQCKCK D9D9DTDTDJDJDQDQDKDK S2S2S3S3S 4S
4S8S8SJSJ
```

主花色是 H，CT 是队 2，级别是 A，FT 是队 1，级别是 A，庄家是 D。

第 1 圈：

D 发牌：首牌刚好拆成 3 个副牌的拖拉机，S5S5S6S6S7S7，SQSQSKSK，S9S9STST。

A 出牌：对于首牌的第 1 个拖拉机，可以用两个结构来对应：H6H6H7H7H8H8 或者是 HTHTHJHJHQHQ。如果使用后者对应，那么剩下的牌就是 H6H6H7H7H8H8HKHK。无法再对应首牌中剩下的两个拖拉机。所以唯一的拆分结构就是 H6H6H7H7H8H8，HTHTHJHJ，HQHQHKHK，这样就比首牌大。而 B、C 的出牌是和 A 不同花色的副牌。

所以这一圈 A 胜出，牌面分数是 110，而 A 是队 1 的成员，FT 得 110 分。

第 2 圈：

A 出大王，而 B、C 出的是副牌，D 出了一张比大王更小的主牌 SA，所以胜者是 A，FT 得 0 分。

第 3 圈，也是最后一圈：

A 又甩出一个拖拉机加两对：H3H3H4H4H5H5，HAHA，H9H9。B、C、D 都是出了一堆副牌。FT 得分 200，首牌中最长的结构长度为 $w = 6$。

然后就是底牌中是 H2H2DACASABJBJRJ，牌面分数为 0。所以 FT 最终得分是 200 分，直接升 3 级，获胜。

☞**注意：**

升级要大于 A 才算赢家，等于 A 不算。因为逻辑层次比较多，建议在编码时一些基本的过程要边写边测，如判断拖拉机，计算一把牌的得分等。并且主要的逻辑过程要有详细的调试日志，以便迅速地发现并定位问题。

SQL 解析（Interpreting SQL, UVa10757，难度 10）

现需要编写一个 SQL 数据库服务器的查询处理部分。服务器包含多张表。表内有行列，每列有固定的数据类型（数字或者字符串）以及名称。表中的每个格子也有类型，就是所在列的类型。同一张表不会有重名列。表与表也不会重名。

表 3.5 就是一个样例表（第一行包含列名，字符串类型左对齐，数字类型右对齐）：

表 3.5

Account	LastName	FirstName	Balance
1	Ivanov	Petr	2500
2	Petrov	Ivan	2000
3	Ivanov	Ivan	3000

一个查询 SQL 语句是一个字符串，告诉服务器从一张表或者多张表获得全部或者部分数据，形成一张临时表发送给客户端（之后临时表就被销毁）。下面是本题中的 SQL 语法：

```
    query ::= 'SELECT' select 'FROM' f rom possible - where possible - order;
    select ::= '*' | column - list;
    column - list ::= name | name ',' column - list;
    f rom ::= name | inner - f rom 'INNER' 'JOIN' inner - f rom 'ON' name '='
name;
    inner - f rom ::= name | '(' inner - f rom 'INNER' 'JOIN' inner - f rom 'ON'
name '=' name ')'; possible - where ::= empty | 'WHERE' where;
    where ::= where - 2 | where - 2 ('AND' | 'OR') where;
    where - 2 ::= '(' where ')' | 'NOT' where - 2 | value operation value;
    operation ::= '=' | '<' | '>' | '<=' | '>=' | '<>';
    value ::= number | string - constant | name;
    possible - order ::= empty | 'ORDER' 'BY' order - by;
    order - by ::= order - column | order - column ', ' order - by;
    order - column ::= name (empty | 'ASCENDING' | 'DESCENDING');
    empty ::= ;
    number ::= (empty | '+' | '-') . unsigned;
    unsigned ::= digit | digit . unsigned;
    digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
    name ::= letter | name . (letter | digit);
    letter ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k'
| 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x'
| 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K'
| 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X'
| 'Y' | 'Z';
    string - constant ::= '"' . escaped - string . '"';
    escaped - string ::= empty | escaped - symbol . escaped - string;
    escaped - symbol ::= digit | letter | special - symbol | other - symbol;
    special - symbol ::= '\\' | '\"';
    other-symbol::='!'|'#'|'$'|'%'|'&'|'''|'('|')'|'★'|'+'|','|'-'|'.'|'/'|
':'|';'|'<'|'='| '>'|'?'|'@'|'['|']'|'^'|'_'|'`'|'{'|'|'|'}'|'~';
```

两个连续项之间的点（.）表示之间可能没有任何空格。如果两项之间没有点，那么之间可能被一到多个空格、制表符或者换行符分开；两个名字之间如果没有点，那么也可能被空格、制表符和换行符分开。括号用来对项进行分组。除了在字符串常量中外，字母不分大小写。

查询的执行逻辑如下：

- 搜索表是从“FROM…”部分生成的。如果“FROM…”部分只包含一个名称，那么它就是要搜索的表名。要么就是这种形式“from1 INNER JOIN from2 ON name1 = name2”，其中 from1 和 from2 都是 from 语句，从其中首先生成 table1 和 table2，

然后再生成结果表。这里 name1 和 name2 都是 table1 和 table2 里面的列名（保证 table1 和 table2 中没有相同的列名）。搜索表的列是这么生成的：首先一个生成一个行列表(table1 的行 1 + table2 的行 1)，(table1 的行 2 + table2 的行 1)，…，(table1 的行 2 + table2 的行 1)，这里“+”表示拼接。这个列表中列 name1 和列 name2 相等的部分被选择出来形成结果表，而行的顺序保持不变。

举例来说，如果 table1 是上文提到的那张表，并且 table2 如表 3.6 所示。

表 3.6

From	To	Amount
1	2	1000
2	3	2000
3	1	3000
2	1	10

那么查询“from1 INNER JOIN from2 ON Account = From”的结果如表 3.7 所示。

表 3.7

Account	LastName	FirstName	Balance	From	To	Amount
1	Ivanov	Petr	2500	1	2	1000
2	Petrov	Ivan	2000	2	3	2000
2	Petrov	Ivan	2000	2	1	10
3	Ivanov	Ivan	3000	3	1	3000

- ❑ 从搜索表中选择出的列形成结果表。如果没有 WHERE 语句，就选择所有行。否则满足 WHERE 语句的行被选择出来。转义的字符串中，\\被认为是\，\"被认为是"。逻辑和比较运算就是一般的含义；字符串按照字典序比较；被比较的值一定是相同的类型。操作是左结合的，“a AND b OR c”意思是“(a AND b) OR c”。WHERE 语句中的名称都是表的列名。
- ❑ 结果表中的行要进行排序。如果有 ORDER BY 语句，行按照语句中的第一列进行排序；如果两行中的这一列值相同，再按照第二列排序。字符串按照字典序排序。ASCENDING 或者 DESCENDING 放在列表之后表示排序的方向，前者是升序，后者是降序。默认是升序，排序必须是静态的，意思是说两个相等的列的顺序在排序之后必须保持不变。
- ❑ 那些要被输出的列会被选择。如果一个 SELECT 后面是一个星号，那么所有的列选中。否则后面给出要选中的列名；要按照给出的顺序输出所有的列（注意，给出的所有列名可能包含重复，这种情况下，相应的列也要按照顺序重复输出）。

给出表格数据以及 SQL 语句，输出相应格式的结果表。输入输出格式较为复杂，详情请参考原题的描述。

【分析】

首先因为牵涉很多的语法分析，这里假设读者已经实现了“递归下降法”，在《算法

竞赛入门经典（第 2 版）》的 11.1 节中对于这种表达式解析的方法有比较详细的介绍，不熟悉的读者可以仔细阅读一下。

Table 结构可以这么设计：

```
typedef vector<string> Row;
struct Table {
    int M, N;
    vector<string> colNames;                  //列名
    vector<char> colTypes;                    //列的类型
    map<string, int, StrIComp> colIndice;    //列名到列编号的索引，StrIComp 作用
见下文
    vector<Row> rows;                         //所有的行
}
```

读取输入的 SQL 语句之后，首先是要对其进行切分，分成 select 的列 cols，from 部分的单词，where 部分的单词以及 order 部分的单词。需要注意的是，题目的不同测试数据之间是用空行分割，需要整行读取，再进行分割。解析时需注意某些表名或者列名可能为 WHERE、FROM、ON、AND 等关键字。

首先是对 from 进行解析执行，拿到所有的单词 fromTokens 之后，使用递归下降法对其进行解释。首先得到 INNER JOIN 和 ON 的位置，然后递归调用解析过程，得到 INNER JOIN 左右两边的语句执行结果 table1 和 table2，然后再使用 ON 后面指定的条件对两个表进行拼接。拼接时，可以首先使用第二个列名 name2 对 table2 的 n2 列建立一个索引，使用 map<string, vector<int>>即可。这个索引包含所有的 n2 列的 value 以及对应的所有行号。然后遍历 table1 的每一行，根据刚刚建立的索引得到对应 table2 中的行号之后拼接，这样就不需要对 table2 的每一行进行遍历。

From 部分的表格生成之后，就需要使用 where 部分的语句对其进行过滤，where 部分的语句，首先对字符串进行处理，将其中的转义字符进行处理，得到所有的单词。之后可以使用递归下降法对其进行解析并且构造表达式树，解析时，首先找到最右边的 ON 或者 AND 的位置，然后把这个位置的两侧表达式分别解析成响应的表达式树。如果找不到 ON 或者 AND，但是可以找到其他运算符，那么使用其他运算符作为根结点返回即可，如果开头有 NOT 要在 ExpNode 中记录一下。注意输入案例里面会有一些表的列名为 AND 或者 OR，并且在 where 条件中使用了，需要注意判断。

表达式的树结点可以使用如下的定义：

```
struct ExpNode{
    string name;
    bool isNot;                                   //是否有 NOT
    ExpNode *left, *right;
    bool eval(Table *pt, Row& row)                //对指定表的指定行进行判断
    string getValue(Table *pt, Row& row)          //叶子结点求值
}
```

在解析出表达式树之后，可以使用表达式树对 From 部分生成的表格行进行过滤，不要在原表中就地过滤，更好更快的方法是将指向符合条件的行的指针存储下来。

最后执行 ORDER BY 时，可以使用 STL 的 stable_sort，然后传入一个比较器作为行指针之间的比较函数：

```
stable_sort(resultRows.begin(), resultRows.end(), RowComp());
```

RowComp 就是一个 STL 里面的 Functor：

```
struct RowComp {
    bool operator() (const PRow&lhs, const PRow& rhs) const;
//对两行按照 ORDER BY 后面输入的列依次进行比较
};
```

需要注意的是，题目中的 Table 名称，以及列名都是大小写无关的，可以将比较逻辑封装成一个 Functor，在各个索引用的 map 中使用：

```
struct StrIComp{
    int strcasecmp(const char *s, const char *t) const {
        while (*s && *t) {
           if (toupper(*s) != toupper(*t)) return toupper(*s) < toupper(*t) ?
-1 : 1;
            s++; t++;
        }
        if (*s == 0 && *t == 0) return 0;
        return toupper(*s) < toupper(*t) ? -1 : 1;
    }

   bool operator()(const string& lhs, const string& rhs) const {
        return strcasecmp(lhs.c_str(), rhs.c_str()) == -1;
    }

    bool eq(const string& lhs, const string& rhs) {
        return strcasecmp(lhs.c_str(), rhs.c_str()) == 0;
    }
};
```

在这个比较逻辑中初学者比较容易犯的错误是在比较两个字符串之前，先 copy 出来然后转成大写再做比较，这样在实际的比较算法执行之前就牵涉大量的内存分配以及 copy，在这个题目中，因为有大量的在 map 中取列名的操作，这样就会大幅度增加代码的运行时间，在笔者的机器上测试时将近有 40 倍的差异。

安全部门（Department, ACM/ICPC Europe - Central 1995, LA5519，难度 10）

安全部门的总部大楼有好多层，每层都有标着 xxyy（0 < xx; yy≤10）格式编号的房间，xx,yy 分别代表楼层以及房间号。楼里装了一个链斗式升降电梯，这种电梯由一系列的隔间连接而成，然后不停地循环移动（参见图 3.14），由于这个特点，乘客只能在特定的时间

点进去（这个题中是每整 5 秒才可以进去，如 10:00:05、10:01:10 等，这一点非常关键）。


图 3.14

一天之中有很多人需要访问大楼，每个人都要访问若干个房间，并且在其中停留一段时间。每个房间或电梯间在同一个时间点只能有一个人。所有人肯定可在一天之内完成计划的访问。每个人从 1 楼进入，通过接待之后开始按计划访问房间。总是按照房间编号的升序进行访问。每个人都有唯一编码，级别越高编码越小。

如果多个人想要进入同一个房间或者电梯隔间，那么就需要按照编号进行排队，编号越小的越靠前。一个人访问完最后一个房间之后，就可以离开大楼；如果不在一层，还要先乘电梯到第一层。

在大楼中做特定的移动，所需要的时间都是固定的（参见原题）。现在需要根据每个人的编码信息，进入时间以及访问计划，模拟输出每个人每一段的时间点以及长度（格式要求参见原题）。

【分析】

首先，这种时间事件模拟的问题，需要用一个优先级队列来管理所有时间点上的事件。对于每个房间，以及大楼中的唯一电梯，都要建立一个优先级队列来对人员按照编码大小进行排序。

```
struct Event {
    EventType type;
    int id, time;
    bool operator<(const Event& e) const {
         return time > e.time || (time == e.time && id > e.id);
     }
    Event(int i, int t, EventType et) : id(i), time(t), type(et) {}
};
```

全局事件队列中都是一系列的事件，按照时间然后是人员的编码进行排序。事件还分为 5 种类型：

```
enum EventType{
   AOutRoom = 0,              //人员从房间出来
   AOutElev,                  //人从电梯中出来
   AReachElev,                //人到达电梯口
   AReachRoom,                //人到达房间口
   ElevNext                   //电梯到达下一个时间点可以进入
};
```

然后对 5 种事件的后续操作进行模拟。

需要注意的是，因为要对人的每一段状态进行输出，还需要记录每个人的当前状态，并且在相应事件发生时改变状态，并输出上次的状态以及持续时间。



有两点需要注意：

（1）如果人员到达电梯门口的时间不是 5 秒的整数倍，需要等到下一个 5 秒整数倍。

（2）时间可以转换成整数，也就是 0:0:0 到这个时间的秒数来计算，这样方便统一处理。

3.3　动 态 规 划

我们一起看电影（Let's Go to the Movies, ACM/ICPC Africa and the Middle East - Africa and Arab - 2007/2008, LA4090，难度 3）

电影院卖两种电影票：单张票只能一人看，家庭套票可以让一个人带着他的孩子（但不包括孩子的孩子）一起看。套票比单票贵，有时可能贵 5 倍。

给出一个家庭的结构，要挑选一种最经济的购票方案。如图 3.15 所示的结构中，有 4 种购票方案：

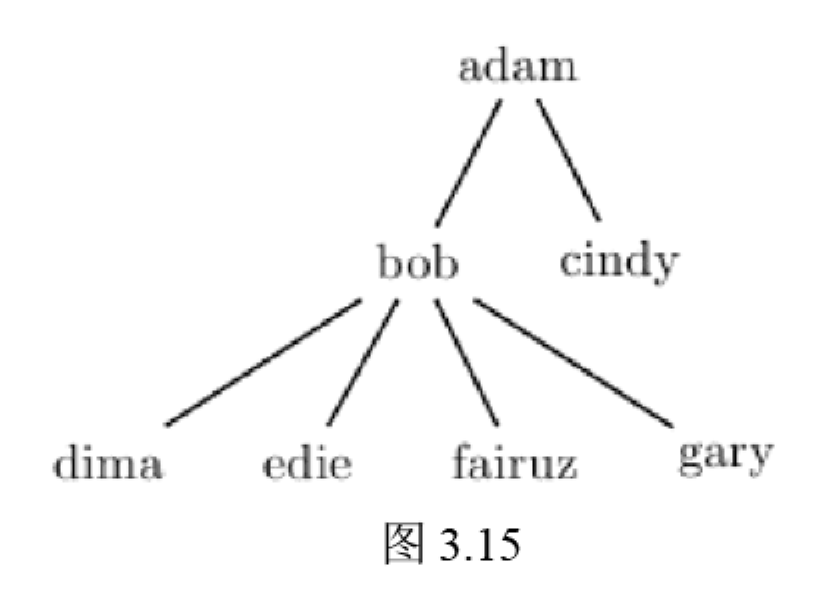


图 3.15

（1）7 张单票。

（2）两张套票。

（3）一张套票（给 adam、bob、cindy）且其他人单票。

（4）一张套票（给 bob 和他的孩子）且其他人单票。

给出套票价格 F 和单票价格 S 以及家庭的结构。计算出一个最优的购票方案，并输出其票价之和。如果问题有多解，输出票数最少的那个。

【分析】

显然，本题中的模型形成一棵树，对于树上的每一个结点 u，记 dp(u,f)为以 u 为根结点的子树的最优方案，其中 f = 0 或 1，表示 u 是否已被套票覆盖。其返回值类型定义如下：

```
typedef long long LL;
struct Ans{                                    //购票方案
    int NS, NF;                                //单票个数，套票个数
    LL T;                                      //购票费用
    Ans(int ns = 0, int nf = 0, LL t = 0) :NS(ns), NF(nf), T(t){}
    void init(int ns = 0, int nf = 0, LL t = 0) { NS = ns, NF = nf, T = t; }
    bool operator<(const Ans& a) const {       //比较规则
```

```
        if (T != a.T) return T < a.T;
        return (NF+NS < a.NF+a.NS);
    }
    Ans& operator+=(const Ans& a){ NS += a.NS, NF += a.NF, T += a.T; return
*this; }
};
```

对于每个结点 u，状态转移方法如下：

（1）u 是叶子结点，v 是 u 的子结点。

① $f=1$，则 u 被套票覆盖，无须买票，$\mathrm{dp}(u,f)=\mathrm{Ans}(0,0,0)$。

② $f=0$，则需要给 u 买票，可以买单票或者套票，$\mathrm{dp}(u,f)=\min(\mathrm{Ans}(1,0,S),\mathrm{Ans}(0,1,F))$。

（2）u 不是叶子结点。

① 首先考虑给 u 买套票和单票两种情况。

- 套票：$\mathrm{dp}(u,f)=\min(\mathrm{dp}(u,f), \mathrm{Ans}(0,1,F)+\sum_v \mathrm{dp}(v,1))$。
- 单票：$\mathrm{dp}(u,f)=\min(\mathrm{dp}(u,f), \mathrm{Ans}(1,0,S)+\sum_v \mathrm{dp}(v,0))$。

② 如果 $f=1$ 则还要考虑，不给 u 买票的情况：$\mathrm{dp}(u,f)=\min(\mathrm{dp}(u,f), \mathrm{Ans}(0,0,0)+\sum_v \mathrm{dp}(v,0))$。

算法的时间复杂度为 $O(n^2)$，主程序（C++11）如下：

```
using namespace std;
typedef vector<int> IVec;
typedef vector<string> SVec;
const int MAXN = 100000 + 4;
int S, F, n;
IVec G[MAXN], roots;
SVec adj[MAXN], words;
map<string, int> indice;

typedef long long LL;
Ans DP[MAXN][2];

const Ans& dp(int u, int f) {
//以u为根结点的子树，是否被套票覆盖?（f = 1,0）对应的最优结果
    Ans& d = DP[u][f];
    if (d.T != -1) return d;
    d.init();

    const IVec& A = G[u]; bool leaf = A.empty(); //u的邻居
    if (leaf) { //u是叶子
        //没被套票覆盖，必须买单票或是套票，否则无须买票
        if (!f) d = min(Ans(1,0,S), Ans(0,1,F));
```

```
            return d;
        }

        //所有孩子 v 的 dp(v,f)之和
        auto sumDP = [&A](int f, Ans dt){ for(auto a : A) dt += dp(a,f); return
dt; };
        //套票或单票
        d = min(sumDP(1,Ans(0,1,F)), sumDP(0,Ans(1,0,S)));
        if(f) d = min(sumDP(0,Ans()), d);                //还可以选择不买
        return d;
    }

    int main() {
        string l, w;
        getline(cin, l);
        for(int t = 1;;t++) {
            assert(isdigit(l[0]));
            stringstream si(l);
            assert(si>>S>>F);
            if (!S && !F) break;
            words.clear(), indice.clear();

            n = 0;
            set<int> cRoots;                             //所有树根
            while (true) {
                getline(cin, l);
                if (isdigit(l[0])) break;
                stringstream ss(l); ss>>w;
                words.push_back(w); indice[w] = n;      //w 是一个父结点
                adj[n].clear(); while (ss >> w) adj[n].push_back(w);
                cRoots.insert(n++);
            }

            _for(i, 0, n){ //建图
                IVec& g = G[i]; g.clear();
                for(auto w : adj[i]){
                    bool leaf = !indice.count(w);       //w 是叶子结点
                    if (leaf) adj[n].clear();
                    int v = leaf ? n++ : indice[w];     //w 对应的结点编号
                    g.push_back(v), cRoots.erase(v);    //v 不是树根
                }
            }
```

```
        roots.assign(cRoots.begin(), cRoots.end());
        Ans ans;
        memset(DP, -1, sizeof(DP));

        for(auto r : roots) ans += dp(r, 0);
        cout << t << ". " << ans.NS << " " << ans.NF << " " << ans.T << endl;
    }
    return 0;
}
```

过山车（Roller Coaster, ACM/ICPC North America - Southeast - 2010/2011, LA4870，难度 4）

贝茜很喜欢坐过山车，但是过程中会头晕。一次要经过 N（$1 \leqslant N \leqslant 1000$）段。一开始头晕和快乐指数都是 0。对于过山车的每一段，贝茜可以选择睁开眼或者一直闭眼。如果睁开，快乐指数增加这一段对应的一个值 F（$1 \leqslant F \leqslant 20$），头晕指数也会增加这一段对应的值 D（$1 \leqslant D \leqslant 500$）。如果闭着眼，总快乐指数不变，但是头晕指数减少 K（$1 \leqslant K \leqslant 500$），头晕指数不会减到 0 以下。如果在任何点上头晕指数超过 L（$1 \leqslant L \leqslant 300000$），就会生病。计算在不生病的前提下贝茜能获得的最高快乐指数。

【分析】

快乐指数的上限是 20*1000，总共 N（$N \leqslant 1000$）段。可以考虑把当前的位置和快乐指数一起作为一个状态来考虑。记 $d(i,f)$为经过前 i 段之后，快乐指数为 f 时最小的头晕指数。

则状态转移过程如下：

（1）如果选择第 i 段闭眼，则 $d(i,f)=\min(d(i,f), \max(0, d(i-1,f)-K))$。

（2）如果 $f \geqslant F[i]$并且 $d(i-1,f-F[i])+D[i] \leqslant L$，则说明第 i 段可以选择睁眼：$d(i,f)=\min(d(i,f), d(i-1,f-F[i]) + D[i])$。

问题的解就是符合 $d(N,f) \leqslant L$ 的最大的 f。时间复杂度为 $O(N^2*\max(F))$。

完整程序（C++11）如下：

```
using namespace std;

const int MAXN = 1000+4, MAXF = 1000*20, INF = 1000000007;
int DP[MAXN][MAXF], F[MAXN], D[MAXN], N,K,L;
int main() {
   while(cin>>N>>K>>L && (N+K+L)){
      int ans = 0, FF = 0;
      _rep(i, 1, N) cin>>F[i]>>D[i], FF += F[i];
      _rep(i, 0, N) _rep(f, 0, FF) DP[i][f] = INF;
      DP[0][0] = 0;
      _rep(i, 1, N) _rep(f, 0, FF){
         int& d = DP[i][f];
         d = min(d, max(0, DP[i-1][f]-K));              //第 i 段闭眼
         if(f >= F[i] && DP[i-1][f-F[i]] + D[i] <= L)   //第 i 段睁眼
```



```
            d = min(d, DP[i-1][f-F[i]] + D[i]);
        }
        _rep(f, 0, FF) if(DP[N][f] <= L) ans = f;
        for(int f = FF; f >= 0; f--)
            if(DP[N][f] <= L) { ans = f; break;}
        cout<<ans<<endl;
    }
}
```

外太空侵略者（Outer space invaders, ACM/ICPC Europe - Central 2014, LA6938，难度 8）

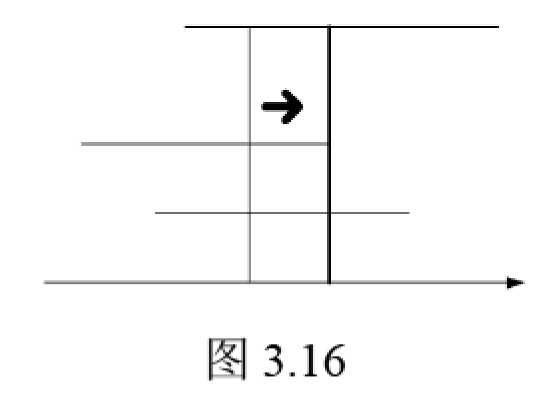
外星人侵略地球了，其中 n（$1 \leqslant n \leqslant 300$）个要攻击你。第 i 个在时间点 a_i 出现在和你距离 d_i 的地方。在时间点 b_i 之前，你必须对它开火，否则它会开火干掉你。其中，$1 \leqslant a_i < b_i \leqslant 10000$；$1 \leqslant d_i \leqslant 10000$。第 i 个外星人在时间点 b_i 之前是不会攻击你的。

你手上有一个个武器，可以设置到任意能量等级。如果以能级 R 发射，就会把距离你在 R 以及更近的外星人全部干掉，同时需要耗费 R 个能量单元。计算在不被杀掉的前提下，杀掉所有的外星人最少需要多少个能量单元。

【分析】

遇到时间相关的问题，关键是把时间看成一个维度，而本题中另外一个维度就是距离。因此可以抽象出如下模型：在二维坐标系中，给出 n 个水平的线段，其中第 i 个线段的左右端点坐标分别是(a_i, d_i)和(b_i, d_i)，然后需要绘制一些起点在 x 轴上的垂直线段，使得所有的水平线段都和某个垂直线段有公共点。同时要求垂直线段长度之和最小。

任意最优解中，如果垂直线段 H 和水平线段的交点不是某个水平线段的端点，则将 H 移到与其相交的所有水平线段的最近的端点，因为其高度不变，且相交的水平线段不变，所以一定仍然是最优解（如图 3.16 所示）。因此只考虑经过线段的左右端点的垂直线段。


图 3.16

把所有端点的 x 坐标放在一起递增排序去重，得到一个集合 p，注意 p 中也要加入$-$INF 和 INF 两个端点，这样把所有的输入线段的端点包含在一个开区间内。记 p 的元素个数为 m。记 $F(i,j)$ 为要穿越水平开区间(p_i,p_j)中的所有水平线段（$p_i<a_k,b_k<p_j$）所需要绘制的垂直线段的长度之和的最小值。对于区间(i,j)来说，考虑其中高度最高的水平线段 h（若有多个，则取最左边的），那么穿过它的垂直线段显然只需要 1 个即可。对这 1 个垂直线段穿过的水平端点编号 x 进行决策（参见图 3.17 中的虚线），则显然有 $F(i,j) = \min(d_h+F(i,x)+F(x,j))$。

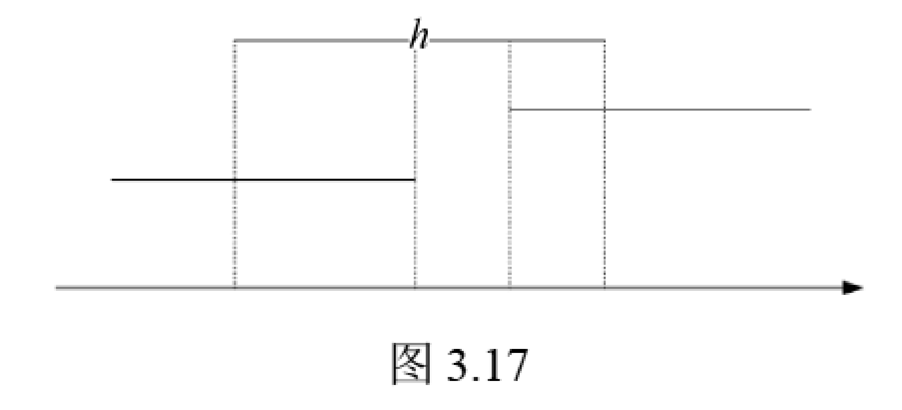


图 3.17

则本题所求的解就是 $F(0,m-1)$。算法的时间复杂度上限是 $O(n^3)$。完整程序（C++11）

如下：

```
    using namespace std;
    int lowerBoundIdx(const vector<int>& v, int x){ return lower_bound(v.begin(),
v.end(), x) - v.begin(); }
    const int N = 610, INF = 0x3f3f3f3f;
    int main(){
        int T, H[N], A[N], B[N], n, f[N][N]; cin>>T;
        vector<int> p;
        while (T --) {
            cin>>n, p.clear();
            _for(i, 0, n) cin>>A[i]>>B[i]>>H[i], p.push_back(A[i]), p.push_back
(B[i]);
            p.push_back(-INF); p.push_back(INF); //增加无穷远的两个点作为边界
            sort(p.begin(), p.end());
            p.erase(unique(p.begin(), p.end()), p.end());
            int k = p.size();
            _for (sl, 1, k) _for(i, 0, k - sl) { //区间长度 sl 遍历
                int j = i + sl, hst = -1; // hst -> 最高的那个区间
                _for (q, 0, n)
                    if (p[i] < A[q] && B[q] < p[j] && (hst == -1 || H[hst] < H[q]))
hst = q;
                int& df = f[i][j]; df = 0;
                if(hst != -1){
                    df = INF;
                    int l = lowerBoundIdx(p, A[hst]), r = lowerBoundIdx(p,
B[hst]);
                  //hst 左右两端对应的 idx
                    _rep (d, l, r) df = min(df, H[hst] + f[i][d] + f[d][j]);
                }
            }
            printf("%d\n", f[0][k - 1]);
        }
        return 0;
    }
```

本题解思路参考了 http://blog.csdn.net/u012647218/article/details/42148639。

3.4 组 合 递 推

我讲鲸鱼的语言（I Speak Whales, ACM/ICPC Africa and the Middle East - Africa and Arab - 2008/2009, LA4369，难度 3）

Walsh 矩阵是满足以下条件的方阵：

（1）行列数都是 2 的幂。

（2）每一项都是±1。

（3）任意不同的两行（或两列）的点积都是 0。

其中前 3 个 Walsh 矩阵是：$W_1=[1]$，$W_2=\begin{array}{c|c} 1 & 1 \\ \hline 1 & -1 \end{array}$，$W_4=\left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{array}\right]$。大小为 2^{N+1} 的 Walsh 矩阵可以使用 4 个 2^N 的矩阵来构造，其中右下角的那个要对其进行翻转：$W_{2^{N+1}}=\left[\begin{array}{c|c} W_{2^N} & W_{2^N} \\ \hline W_{2^N} & -W_{2^N} \end{array}\right]$。给出 $N(0\leqslant N\leqslant 60)$、$R(0\leqslant R<2^N)$、S、$E(0\leqslant S\leqslant E\leqslant 2N, E-S\leqslant 10000)$ 4 个整数，计算大小为 2^N 的矩阵中第 R 行上第 S 到 E 列的值之和。

本题中行列编号都是从 0 开始计算。

【分析】

2^N 的矩阵可以看作分为 4 个大小为 2^{N-1} 的子方阵，把输入的区域分成分别位于 4 个子方阵的 4 个部分，然后递归计算 4 个部分，最终求和即可。当 N=0 时结果为 1。

疯狂的考试（Deranged Exams, ACM/ICPC North America - Greater NY 2013, LA6469，难度 4）

给出 N（$1\leqslant N\leqslant 17$）个名词，以及 N 个对应的解释。学生需要为每个名词找到正确的解释。定义 $S(N,k)$（$0\leqslant k\leqslant N$）为完全随机回答，至少前 k 个答案是错误的回答方案个数。输入 N,k 计算 $S(N,k)$。

【分析】

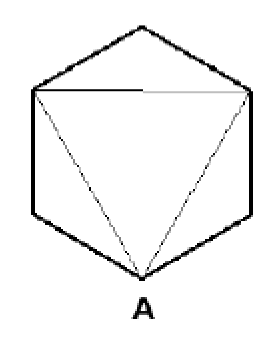
求至少前 k 个是错误的方案数比较难，可以反过来直接求总的方案个数（$N!$）减去其反例，反例也就是前 k 个答案正确的方案个数。定义 A_j 为所有回答方案中问题 j 答对的方案的集合，那么 $S_{N,k}=N!-|A_1\cup A_2\cup\cdots\cup A_k|$。根据容斥原理有：

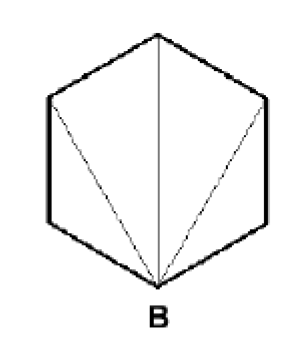
$$|A_1\cup A_2\cup\cdots\cup A_k|=\sum_{i=1}^{k}(-1)^{i+1}\sum_{1\leqslant j_1<\cdots<j_i\leqslant k}|A_{j_1}\cap A_{j_2}\cap\cdots\cap A_{j_i}|$$

对于特定的问题集合 $j_1,j_2,\cdots j_i$，$|A_{j_1}\cap A_{j_2}\cap\cdots\cap A_{j_i}|$ 就是问题 $j_1,j_2,\cdots j_i$ 全部回答对的方案个数，对于其他 $N-i$ 个问题的答案没有限制，这个方案数就是$(N-i)!$。从 k 个问题中选择 i 个问题的方案数为 C_k^i，故有 $\sum_{1\leqslant j_1<\cdots<j_i\leqslant k}|A_{j_1}\cap A_{j_2}\cap\cdots\cap A_{j_i}|=C_k^i(N-i)!$。所以有 $S_{N,k}=N!-\sum_{i=1}^{k}(-1)^{i+1}C_k^i(N-i)!=N!+\sum_{i=1}^{k}(-1)^i C_k^i(N-i)!$。可以提前预处理出所有的 $N!$ 和 C_k^i，然后对于每一组输入的 N,K 递推即可。

三角剖分的三角数目序列（Triangle Count Sequences of Polygon Triangulations, Regionals 2013 North America - Greater NY, LA6471，难度 4）

一个 n 边形的三角剖分是用 $n-3$ 条不相交的内对角线，把一个 n 边形划分成 $n-2$ 个三角形，如图 3.18 所示。





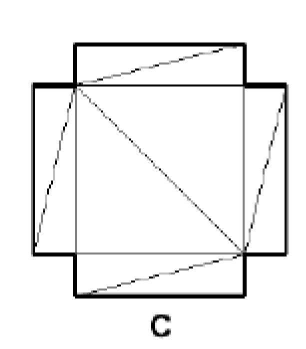


图 3.18

三角剖分的三角数目序列指的是，依次列出与每个顶点相连的三角形数目形成的序列。给出一个长度 N 的三角数目序列，判断是否为一个合法的 N 边形的三角数目序列。如果合法，列出其对应的剖分中的所有三角形的顶点编号，按照字典序排列。

【分析】

本题思路和《算法竞赛入门经典（第 2 版）》中 277 页的例题“最优三角剖分”类似。观察一个 n 边形的剖分，因为是不相交的对角线，可以肯定任意一条这样的对角线都可以把这个 n 边形分成两个更小的多边形。但是依然不好进行递归处理，进一步观察发现必然存在三角形是由一个顶点和其相邻的两个顶点连接而成，这个顶点对应的三角形数目就是 1。如果去掉这个三角形就变成一个 $n-1$ 边形，这样就方便递归处理。具体来说，对于一个长度为 N 的序列可以按照如下顺序处理：

（1）$N<3$，说明输入非法。

（2）$N=3$，就说明是三角形，则要求序列必须是{1,1,1}，否则非法。

（3）$N>3$，记其中一个三角形数目为 1 的顶点编号为 p，其左右顶点为 pl、pr。则 pl 和 pr 对应的数目必须大于 1，否则非法。把 p、pl、pr 对应的三角形从多边形中删掉并记录下来，对形成的 $N-1$ 边形进行递归处理即可。

算法的时间复杂度为 $O(N^2)$。需要注意的是，记录三角形时，每个三角形都使用 vector<int>记录顶点编号，对 3 个数排序之后记录到一个 vector<vector<int>>中，然后可以调用 STL 的排序算法（sort）按照字典序对所有的三角形按照字典序排序输出。

岳飞的战斗（Yue Fei's Battle, ACM/ICPC Asia – Guangzhou 2014, LA7078，难度 8）

岳飞在抗金斗争中最重要的胜利是朱仙镇大捷。当时他要部署一些营房，营房之间用道路连通。为节省成本，希望道路越少越好。而且不能有一个营房过于重要，否则会被敌人攻击。也就是说不存在连接超过 3 条以上道路的营房。根据他的战争理论，需要营房之间的路径的长度最大值刚好是 K。注意，路径的长度指的是其路过的营房的个数，并且长度为 K 的最长路径可能有多条。

岳飞希望知道，在上述条件下有多少种方式部署营房和道路。所有的营房等价，并且建造的营房数量可以认为是无限的。例如，$K=3$，有两种方式部署（如图 3.19 所示）。如果 $K=4$，有 3 种方式（如图 3.20 所示）。图 3.19 和图 3.20 中的点表示营房，线段表示道路。

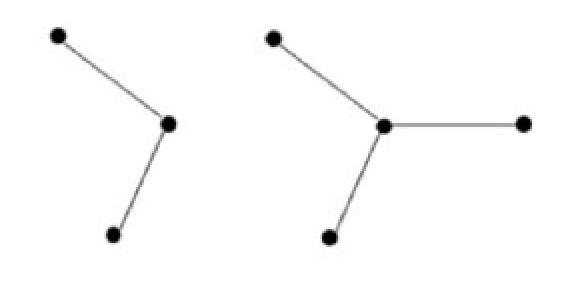


图 3.19

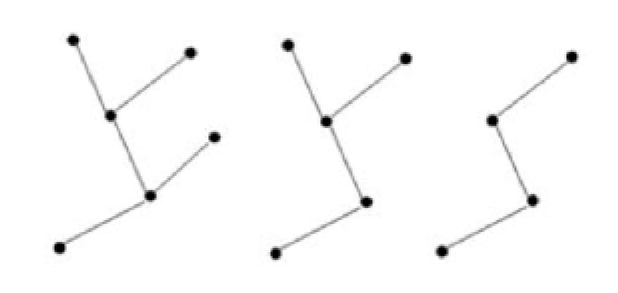


图 3.20

输入 K（$1 \leqslant K \leqslant 100000$），表示最长路径的值。计算共有多少种不同的部署方式，输出其模 1000000007 的值。

【分析】

参考本书中第 2 章习题 11-16 题解的证明思路，不难证明所有的最长路径必然经过同一个点（k 为奇数）或者同一条边（k 为偶数）。这个点（边）是唯一的，把这个点（边）删除就得到一些二叉树。记这个点（边）为结点（边）中点。

记 D_i 为深度 i 的异构二叉树数量，S_i 为深度不超过 i 的异构二叉树数量。深度 i 的二叉树的两颗子树有 3 种情况：

（1）一个深度为 $i-1$，另外一个小于 $i-1$，则有 $D_{i-1}*S_{i-2}$ 种。

（2）深度相同，都是 $i-1$，但结构不同，有$(D_{i-1}*(D_{i-1}-1))/2$ 种。

（3）深度和结构相同，有 D_{i-1} 种。

由此可得：$D_i = D_{i-1} + (D_{i-1}*(D_{i-1}-1)) / 2 + D_{i-1}* S_{i-2}$。

回到本题，记 $n = K/2$，当 K 为偶数，可以认为是深度都是 n 的两个二叉树的根结点用一条边（边中点）连在一起。则图的结构按照两个二叉树的结构异同讨论有两种情况。

（1）两树结构不同：$D_n*(D_n-1)/2$。

（2）两树结构相同：D_n。

所以问题的解是 $D_n+D_n*(D_n-1)/2$。

当 K 为奇数时，可以认为是结点中点连接了 3 个分支二叉树，至少有 2 个深度为 n。

（1）有 1 个分支深度小于 n 的方案数：$S_{n-1}*(D_n*(D_n-1)/2 + D_n)$。

（2）3 个分支深度都是 n，按照 3 个分支结构的异同讨论。

① 3 个结构相同的方案数：D_n。

② 其中 2 个相同的方案数：$D_n*(D_n -1)$。

③ 3 个互不相同的方案数：$D_n*(D_n-1)*(D_n-2)/6$。

问题的解是 $S_{n-1}*(D_n*(D_n-1)/2+D_n) + D_n*(D_n-1) + D_n*(D_n-1)*(D_n-2)/6$。

☞**注意：**

（1）本题中的 D 值和 S 值都需要用 long long，以免数据溢出，而且计算过程中需要每一步的计算结果都求其 Mod。但是其中有除法，必须变成乘法，也就是事先求出 2,6 关于模 Mod 的逆，然后应用在乘法中。求逆的过程请参考《算法竞赛入门经典（第 2 版）》中的 10.1.4 节。

（2）D_0 和 S_0 也应该等于 1，而不能是 0，因为没有结点也算一种方案。

本题思路参考了 http://www.cnblogs.com/crackpotisback/p/4801279.html。

塔防游戏（Tower Defense, ACM/ICPC Asia – Hangzhou 2013, LA6463，难度 8）

DRD 设计了一种塔防游戏。在一个 N 行×M 列的网格中（$1 \leqslant N, M \leqslant 200$），每个方格中只能放一个塔，整个网格中至少要放一个塔。塔分重塔和轻塔两类，每种塔都能攻击到在同一行或者同一列中的其他塔。轻塔不能被攻击，但是重塔可以被最多一个塔攻击。有 P 个重塔，Q 个轻塔（$0 \leqslant P,Q \leqslant 200$），要放到网格中，可以不全放进去，计算有多少种放置方案。输出其模 10^9+7 的值。

【分析】

由于重塔最多能被另一个重塔攻击，所以比较麻烦的是包含两个重塔的情况。可以首先对包含两个重塔的行列进行讨论。

依次对包含两个重塔的行数 i 和列数 j 进行遍历，i 和 j 确定后，剩下未放塔的行数就是 $r=N-(i+2j)$，列数 $c=M-(2i+j)$，剩下的重塔个数就是 $P-2(i+j)$。在 N 行 M 列的棋盘中选择 i 行放两个重塔，还要再选择 $2i$ 个列作为放重塔的位置，方案个数就是 $C_N^iC_M^{2i}$。选择 $2i$ 列之后，将其分成 i 组，每一组对应每一行中 2 个重塔的位置。首先对 $2i$ 个位置进行排列选择，再对 i 个组的组内进行去重，所以方案数为 $\frac{(2i)!}{2^i}$。选择 j 列的推导过程类似。

接下来就是独占一行和一列的塔，首先对 $r\times c$ 棋盘中塔的总个数 k 进行决策，k 确定之后，再考虑其中重塔的个数 p，之后就可以得到 $r\times c$ 棋盘中放置 p 和重塔和 $k-p$ 个轻塔的方案数：$C_r^kC_c^kC_k^pk!$。其中 k!表示要对每一行的塔所在的列进行选择，选择 k 次。注意最终求出答案之后，还要减去 1，也就是一个塔都不放的情况。

关于本题中模求逆，记 $M=10^9+7$, 则 $\frac{1}{2^i}\equiv\frac{(M+1)^i}{2^i}\equiv\left(\frac{M+1}{2}\right)^i(\bmod M)\equiv(5\times10^8+4)^i$ $(\bmod M)$。则模的除法就变成乘法。完整程序如下：

```
using namespace std;

typedef long long LL;
const int MAXN = 200+4;
const LL MOD = 1000000007, R = 500000004;
//组合数，阶乘，组合数求和
LL C[MAXN][MAXN], Fac[MAXN*2], CS[MAXN][MAXN];
LL G2[MAXN];//G2[p] = (2p)!/(2^p)表示，1~2p 的 2p 个数字，被分成 p 个都包含 2 个
数字的组，所有的分组方案数

void prepare(){
    Fac[0] = 1;
    _for (i, 1, 2* MAXN) Fac[i] = (Fac[i-1]*i) % MOD; //阶乘
    fill_n(G2, MAXN, 0);
    G2[0] = 1, G2[1] = 1;
    LL rp = R; //1/2^p ≡ ((MOD+1)/2)^p ≡(500000004)^p (mod MOD)
    _for (i, 2, MAXN) {
        rp = (rp * R) % MOD;
        G2[i] = (Fac[2*i]*rp) % MOD;
    }
    memset(C, 0, sizeof(C));
    C[0][0] = 1;
    _for (i, 1, MAXN) {
        C[i][0] = 1, C[i][i] = 1;
```

```
      _for (j, 1, i) C[i][j] = (C[i-1][j-1] + C[i-1][j]) %  MOD;
   }
   memset(CS, 0, sizeof(CS));
   _for (i, 0, MAXN) {
      CS[i][0] = 1;
      _for (j, 1, i + 1) CS[i][j] = (CS[i][j-1] + C[i][j]) % MOD;
   }
   return ;
}
//在 r 行*c 列的棋盘里，选择 p 行，2*p 列放置每行两个重塔
LL p2C(int r, int c, int p) {
   return ((C[r][p] * C[c][2*p]) % MOD * G2[p]) % MOD;
}

//z 个塔，最少放 minP 个重塔，最多放 maxP 个重塔的方案数
LL p1C(int k, int minP, int maxP) {
   if (minP > 0)
      return ((CS[k][maxP] - CS[k][minP-1]) % MOD + MOD) % MOD;
   return (CS[k][maxP]) % MOD;
}
int main(){
   int N, M, P, Q, T; cin>>T;
   prepare();
   while (T--) {
      cin>>N>>M>>P>>Q; //行，列，重，轻
      LL ans = 0;

      //i 行，j 列上各 2 个重塔
      _rep(i, 0, N) _rep(j, 0, M) {
         const int r = N-(i+2*j), c = M-(2*i+j), p = P-2*(i+j);
         //剩下 r 行 c 列，p 个重塔以及 Q 个轻塔都只占一行
         if (r<0 || c<0 || p<0) continue;

         //i 行，j 列 d 都有 2 重塔的选择方案数
         LL ijC = (p2C(N, M, i) * p2C(M-2*i, N-i, j)) % MOD;

         //剩下的共要放 k 个塔，都是占 1 行 1 列的
         for (int k = 0; k <= p+Q && k <= min(r,c); k++) {
            //k 个塔中重塔个数的上下限
            int minp = max(0, k-Q), maxp = min(k,p);
            //放置 k 个塔的方案数
            LL pq1c = C[r][k] * C[c][k] % MOD * p1C(k, minp, maxp) % MOD;
            ans = (pq1c * Fac[k] % MOD * ijC % MOD + ans) % MOD;
         }
      }
```

```
        cout<<ans-1<<endl; //要减去什么都不放的情况
    }
    return 0;
}
```

本题解思路参考了 http://www.cnblogs.com/wangsouc/articles/3639137.html。

游戏激活（Activation, ACM/ICPC Asia - Beijing 2011, LA5721，难度 8）

T 是游戏《仙剑奇侠传 5》的粉丝，在买了游戏之后，要在线激活。所有的激活请求形成一个队列。服务端在处理队列顶端的请求时，可能出现 4 种情况：

（1）激活失败，概率为 $p1$，客户端的请求仍在队首，服务端会重试处理。

（2）连接失败，表示发出请求的客户端已经断开，概率为 $p2$。客户端会重新发出一个请求，并且被排到队尾。

（3）激活成功，概率为 $p3$。

（4）服务器宕机，并且队列中的请求全部丢失，概率为 $p4$。之后就不进行处理。

其中（$0\leqslant p1, p2, p3, p4\leqslant 1$，$p1+p2+p3+p4=1$）。现在等待激活的队列长度为 N，T 排在第 M（$1\leqslant M\leqslant N\leqslant 2000$）位，对于 M 来说就有 3 种事件可能发生：

（1）成功激活。

（2）服务器宕机，且此时队列中排在他前面的不超过 $K-1$ 个人。

（3）服务器宕机，队列中排在他前面的至少有 K（$1\leqslant K$）个人。

现在需要计算第 2 种事件发生的概率（结果保留小数点后 5 位）。

【分析】

记 $D_{i,j}$ 表示队列长度为 i 且 T 排第 j 位时，事件 2 发生的概率。$j = 1$ 时，T 排在第 1 位，可能有 3 种情况导致事件 2：

（1）T 激活失败，重试一次，概率 $p1$。

（2）T 连接失败，回到队尾，概率 $p2$。

（3）服务器宕机，概率 $p4$。

所以可得如下的状态转移方程：$D_{i,1} = D_{i,1}*\mathrm{p1} + D_{i,i}*p2 + p4$。另外，$j\leqslant K$ 时有 $D_{i,j} = D_{i,j}*p1 + D_{i,j-1}*p2 + D_{i-1,j-1}*p3 + p4$。等式后面的几项分别对应：队首激活失败重试；队首客户端连接失败；队首激活成功；服务器宕机。如果 $j>K$，因为不需要考虑服务器宕机的情况，则有 $D_{i,j} = D_{i,j}*p1 + D_{i,j-1}*p2 + D_{i-1,j-1}*p3$。

对上述转移方程进行移项处理，记 $q_2=p2/(1-p1)$，$q_3=p3/(1-p1)$，$q_4=p4/(1-p1)$，则有：

（1）$D_{i,1} = D_{i,i}*q_2 + q_4$。

（2）$j\leqslant K$ 时，$D_{i,j} = D_{i,j-1}*q_2 + D_{i-1,j-1}*q_3 + q_4$。

（3）$j > K$ 时，$D_{i,j} = D_{i,j-1}*q_2 + D_{i-1,j-1}*q_3$。

对于确定的 i，记上述两个公式中，等号右边第一个“+”后面的部分为 C_j，则有 $D_{i,j}=D_{i,j-1}*q_2+C_j$。从小到大递推 i 时，C_j 可以根据 $i-1$ 时的结果提前计算出来。但是有个问题就是 $D_{i,1}$ 是由 $D_{i,i}$ 得来，递推时会出现死循环。为解决此问题，使用上述公式对 $D_{i,i}$ 展开：

$$D_{i,i}=D_{i,1}q_2^{i-1}+C_2q_2^{i-2}+\cdots+C_{i-k}q_2^k+\cdots+C_i=D_{i,i}q_2^i+q_4q_2^{i-1}+\sum_{x=2}^{i}C_{i-x}q_2^{i-x}$$

移项可得：$D_{i,i}=\dfrac{q_4q_2^{i-1}+\sum_{x=2}^{i}C_{i-x}q_2^{i-x}}{1-q_2^i}$。

边界条件是：队列中只有 1 个人，只有 1 种情况导致事件 2 就是服务器宕机：$D_{1,1}=\dfrac{p_4}{1-p_1-p_2}$。这样所有的值就可以推导出来。算法时间复杂度为 $O(n^2)$。

完整程序如下：

```
using namespace std;
const double eps = 1e-10;
double dcmp(double x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
const int MAXN = 2000 + 4;
double D[MAXN][MAXN], C[MAXN];
int main(){
    int N, M, K;
    double p1, p2, p3, p4, q2, q3, q4;
    while (scanf("%d%d%d%lf%lf%lf%lf", &N, &M, &K, &p1, &p2, &p3, &p4) == 7) {
        if (dcmp(p4) == 0) { puts("0.00000"); continue; }
        q2 = p2 / (1 - p1), q3 = p3 / (1 - p1), q4 = p4 / (1 - p1);
        D[1][1] = p4 / (1 - p1 - p2);
        _rep(i, 2, N){
            _rep(j, 2, (i<K?i:K)) C[j] = D[i-1][j-1] * q3 + q4;
            _rep(j, K+1, i) C[j] = D[i-1][j-1] * q3;
            double q2Pow = 1, cs = 0; //q2Pow = q2^i
            for (int j = i; j > 1; j--) cs += q2Pow * C[j], q2Pow *= q2;
            D[i][i] = (cs + q2Pow * q4) / (1 - q2Pow * q2);
            D[i][1] = q2 * D[i][i] + q4;
            _for(j, 2, i) D[i][j] = q2 * D[i][j - 1] + C[j];
        }
        printf("%.5f\n", D[N][M]);
    }
    return 0;
}
```

3.5　图　　论

哲学家的竞争问题（The Dueling Philosophers Problem, ACM/ICPC - North America -Mid-Atlantic USA, LA6195，难度 5）

给出编号为 1～n（1≤n≤1000）篇论文，以及 m（1≤m≤50000）个形如 d,u（1≤u, d≤n，d≠u）的引用关系，表示论文 d 中定义的一个名词在论文 u 中被引用。要对这 n 篇

论文进行重新排序，排序方案满足如下要求：一篇论文引用的名词必须在排在它前面的论文中被定义过。

计算一下有多少种满足要求的方案，如果无解输出 0，如果有唯一解输出 1，如果有多解则输出 2。

【分析】

论文根据引用关系形成一个有向图，本题实际上是求拓扑排序的方案个数的情况（是 0、1 还是大于 1）。使用 BFS 遍历入度为 0 的结点 u，每遍历到一个 u 就作为下一个排序结点加入拓扑序，同时在图中删除 u。更新 u 的孩子 v 的入度之后，把入度为 0 的 v 入队。

遍历过程中如果有多个 u 满足要求，则说明有多种拓扑排序方案。如果图还没遍历完就发现找不到入度为 0 的结点，问题无解。

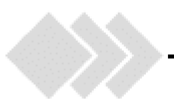
完整程序（C++ 11）如下：

```
using namespace std;
const int MAXN = 1024;
vector<int> G[MAXN];            //有向图
int IND[MAXN];                  //IND[u]是 u 的入度

int main(){
   int n, m, u, d;
   queue<int> q;
   while(cin>>n>>m && n && m){
      _for(i, 0, n) G[i].clear();
      fill_n(IND, n, 0);
      _for(i, 0, m) cin>>d>>u, d--, u--, G[d].push_back(u), IND[u]++;

      int cnt = 0, ans = 1;
      bool flag = false;
      _for(i, 0, n) if(IND[i] == 0) q.push(i);
      while(!q.empty()){
         cnt++;
         if(q.size() > 1) flag = true;
         u = q.front(); q.pop();
         for(int v : G[u]){ IND[v]--; if(IND[v] == 0) q.push(v); }
      }

      if(cnt != n) ans = 0;
      else if(flag) ans = 2;
      cout<<ans<<endl;
   }
   return 0;
}
```



3.6　正则表达式

数字模式（Digit Patterns, UVa12415，难度 10）

R 表达式的构造方法如下：

（1）0, 1, 2, …, 9 以及 0*, 1*, …, 9* 都是 R 表达式。

（2）如果 A 和 B 是 R 表达式，那么(A)、A+B、AB 和(A)*都是。

（3）联合，A+B 匹配所有被 A 或 B 匹配的字符串 s。

（4）串联，AB 匹配所有形为 s_1s_2（s_1 和 s_2 的串联）的字符串，其中 A 匹配 s_1 且 B 匹配 s_2。

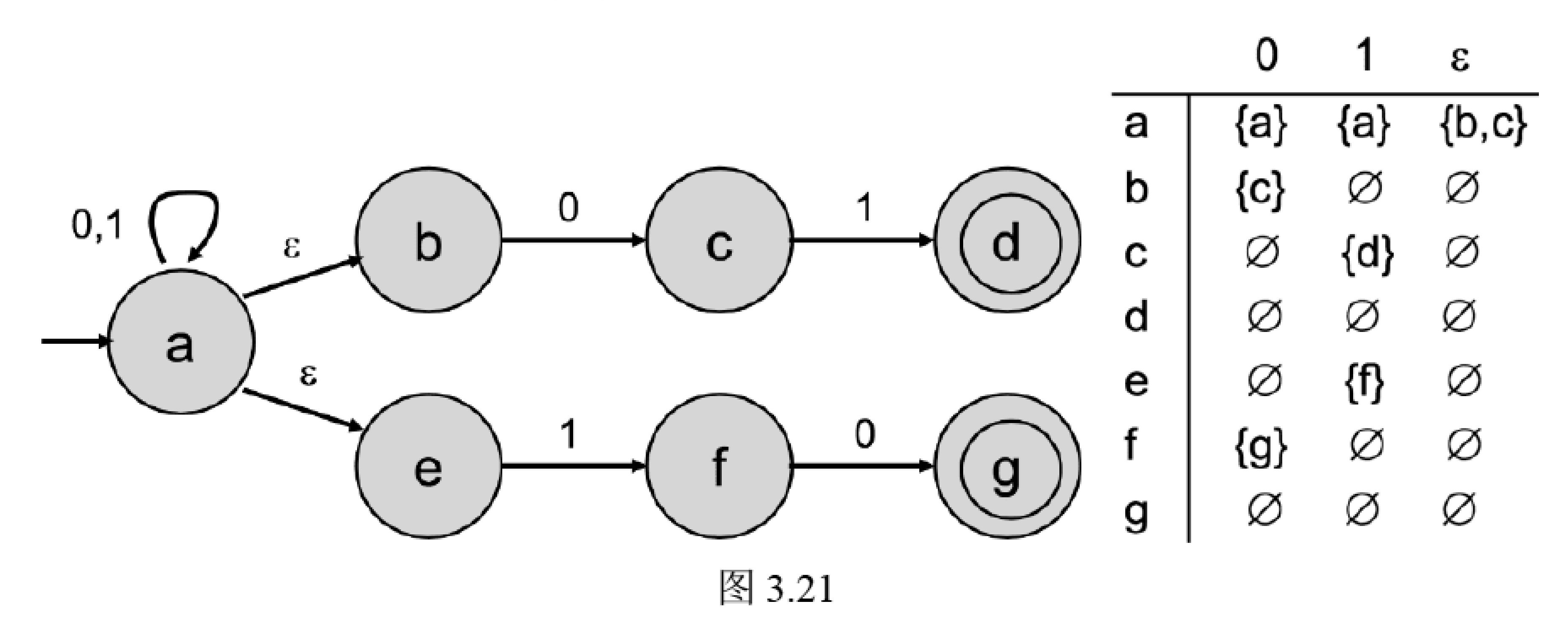
（5）闭包，(A)*匹配所有形为 $s_1s_2s_3$ … s_k（$k\geqslant0$）的字符串，其中 A 可以匹配每个 s_i（注意 s_1, s_2, …不一定相等）。

R 表达式只能用规则 1 和 2 构造。在本题中，R 表达式不匹配空字符串。注意串联的优先级比联合高。所以 11+22 应该解释为 (11)+(22)而不是 1(1+2)2。

给出一个文本 T（长度不超过 10^7）以及仅使用了数字 0～n−1（1≤n≤10）的表达式 R（长度不超过 500），需要找到所有的匹配点 p，使得 R 匹配 T 的[1,p]的子串。例如 R='1(2+3)*4', T = '012345'。只有一个匹配点 5，因为 T 匹配 1234，在位置 5 结束。

【分析】

参考《算法竞赛入门经典（第 2 版）》第 12 章的内容，NFA 和 DFA 的理论模型这里就不做详细介绍。不过要想更好地理解 DFA 和 NFA，更好的办法是结合实际的例子。图 3.21 中 NFA 的 δ 函数实际上可以认为是右侧表所描述的映射。



	0	1	ε
a	{a}	{a}	{b,c}
b	{c}	∅	∅
c	∅	{d}	∅
d	∅	∅	∅
e	∅	{f}	∅
f	{g}	∅	∅
g	∅	∅	∅

图 3.21

使用 NFA 进行状态转移的过程通俗地来讲就是：

（1）每输入一个字符后，跟着图中边的方向能走的都走一步。

（2）随时（输入一个字符前后）沿着 ε 边走一步。

（3）如果在某个阶段，走出了一个预想的状态集合，就认为是匹配成功。

上述的 NFA 对于输入字符串 0010 来说，匹配过程如下：

（1）没有任何初始输入时的状态集合：{a,b,e}。

（2）输入 0 之后：{a,b,e,e}。

（3）0 之后 ：{a,b,e,c}。

（4）1 之后: {a,b,e,d,f}。

（5）0 之后: {a,b,e,c,g}。

因为 g 是一个终态结点，所以匹配成功。根据正则表达式解析并且建立 NFA 的过程请参考《算法竞赛入门经典（第 2 版）》第 12 章的例题。

回到本题，因为匹配的字符串的起点是任意的，所以需要从建完 DFA 之后从 0 到 0 连|Σ|条边，标号分别是|Σ|的所有字母，这样等于每次输入新的字符之后都会有一条路径进行重新匹配。

匹配逻辑如下：假设已经读入字符串 S 的前 i 个字符，则需要求出每个字符转移之后的状态集来查看是否有合法的匹配。可以事先预处理出每一个结点经过每一个字母 ch 之后所能到达的集合，那么每一个转移的复杂度就是 $O(|Q|^2)$，其中|Q|是状态集的大小。但输入字符串长度可能是 10^7，那么这个复杂度是远远不够的。

考虑优化时，很自然地会类比记忆化搜索的思路，对于一个结点集 S，缓存其经过每一个字符 ch 转移之后的结果，用 map 的话，占用存储空间较大，而且做查找的时间也比较长。可以考虑计算其 hash 之后缓存转移的结果集。但是这样的话，提交发现依然是 TLE。

问题出在哪里呢？其实就是每次都要遍历集合计算其 hash，依然有 $10^7*O(|\Sigma|)$的时间复杂度。更好的办法是在匹配的过程中直接使用每个集合的 hash 来建立 DFA。完整程序如下：

```
using namespace std;

typedef unsigned long long LL;
typedef std::vector<int> VI, *PVI;
template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v) {
   bool first = true;
   for(const auto& e : v){
      if(first) first = false; else os <<" ";
      os<<e;
   }
   return os;
}

struct ExpNode{
   enum {A, STAR, OR, CONCAT};
   int type, val;
   ExpNode *l, *r;
   ExpNode(int type, ExpNode *l = nullptr, ExpNode *r = nullptr, int val =
-1)
      : type(type), l(l), r(r), val(val) {};
```

```
    ~ExpNode() {
        if(l) delete l;
        if(r) delete r;
    }
};

ostream& operator<<(ostream& os, ExpNode *pn) {
    if(!pn) return os;
    switch(pn->type) {
        case ExpNode::A:
            os<<(char)(pn->val);
            break;
        case ExpNode::STAR:
            os<<"("<<pn->l<<")*";
            break;
        case ExpNode::OR:
            os<<'('<<pn->l<<')'<<'+'<<'('<<pn->r<<')';
            break;
        case ExpNode::CONCAT:
            os<<pn->l<<pn->r;
            break;
        default:
            assert(false);
    }

    return os;
}

struct RexParser{
    string rex;
    int p, n;
    void skip(char c) { p++; } //for debug purpose
    ExpNode *item() { //(u)* || u closure
        ExpNode *u;
        if(rex[p] == '(')
            skip(rex[p]), u = expr(), skip(')');
        else
            u = new ExpNode(ExpNode::A, nullptr, nullptr, rex[p++]);
        while(rex[p] == '*')
            skip(rex[p]), u = new ExpNode(ExpNode::STAR, u, nullptr);
        return u;
    }

    ExpNode *concat() { //u1u2u3...  concatenation
```

```
        ExpNode *u = item();
        while(p < n && rex[p] != ')' && rex[p] != '+')
            u = new ExpNode(ExpNode::CONCAT, u, item());
        return u;
    }

    ExpNode *expr() {
        ExpNode *u = concat();
        while(rex[p] == '+') {
            skip(rex[p]);
            u = new ExpNode(ExpNode::OR, u, concat());
        }

        return u;
    }

    ExpNode *parse(const string& str) {
        rex = str, n = rex.length(), p = 0;
        return expr();
    }
};

template<int MAXS>
struct NFA{
    struct Transition{
        int ch, next;
        Transition(int ch=0, int next=0):ch(ch),next(next){}
        bool operator<(const Transition& rhs) const {
            if(ch != rhs.ch) return ch < rhs.ch;
            return next < rhs.next;
        }
    };

    int n, MAXA; //num of states, alphabet size
    typedef bitset<MAXS> SSet;
    typedef std::vector<Transition> TVec;
    TVec trans[MAXS];
    vector<SSet> sNextCache[MAXS];

    void add(int s, int t, int c=-1) { trans[s].push_back(Transition(c, t)); }

    void process(ExpNode *u){
         int st = n++, m;
         switch(u->type) {
```



```
        case ExpNode::A:
            add(st, n, u->val);
            break;
        case ExpNode::STAR:
            process(u->l);
            add(st, st+1), add(st, n), add(n-1, st);
            break;
        case ExpNode::OR:
            process(u->l);
            m = n;
            process(u->r);
            add(st, st+1), add(st, m), add(m-1, n), add(n-1, n);
            break;
        case ExpNode::CONCAT:
            add(st, st+1);
            process(u->l);
            add(n-1, n);
            process(u->r);
            add(n-1, n);
            break;
        default:
            assert(false);
    }
    n++;                  //state 'end'
}

void init(const string& rex, int maxa) {
    RexParser rp;
    ExpNode *p = rp.parse(rex);
    n = 0;
    _for(i, 0, MAXS) trans[i].clear();
    process(p);
    MAXA = maxa;
    delete p;
}

VI ss, es;                //starting and ending states
void remove_epsilon() { //remove ε
    VI reachable[MAXS], vis(MAXS, 0);
    _for(i, 0, n){ //BFS to find epsilon-closure for each state
        reachable[i].assign(1, i);
        queue<int> q; q.push(i);
        vis.assign(MAXS, 0), vis[i] = 1;
        while(!q.empty()) {
```

```
                int s = q.front(); q.pop();
                for(const auto& ts : trans[s]){
                    if(ts.ch != -1) continue;
                    int s2 = ts.next;
                    if(vis[s2]) continue;
                    reachable[i].push_back(s2);
                    vis[s2] = 1;
                    q.push(s2);
                }
            }
        }

        ss = reachable[0];
        _for(i, 0, n){ //merge transitions
            set<Transition> tr;
            for(auto& t : trans[i]) {
                if(t.ch == -1) continue;
                for(const auto r : reachable[t.next])
                    tr.insert(Transition(t.ch, r));
            }
            trans[i].assign(tr.begin(), tr.end());
        }

        buildNextCache();
    }

    void buildNextCache(){
        _for(s, 0, n+1){
            auto& sc = sNextCache[s];
            sc.clear(), sc.resize(MAXA);
            for(const auto& t : trans[s])
                if(t.ch != -1) sc[t.ch].set(t.next);
        }
    }

    void delta(int ch, const SSet& from, SSet& to) const { // δ
        to.reset();
        _for(s, 0, n+1) if(from.test(s)) to |= sNextCache[s][ch];
    }
};

template<int MAXS>
ostream& operator<<(ostream& os, const NFA<MAXS>& nfa) {
    os<<"starting: "<<nfa.ss<<", n = "<<nfa.n<<endl;
```



```
    _for(s, 0, MAXS){
        const typename NFA<MAXS>::TVec& ts = nfa.trans[s];
        if(ts.empty()) continue;
        os<<s<<" : ";
        for(auto t : ts){
            os<<", -"<<(t.ch == -1 ? 'e' : (char)(t.ch));
            os<<"->"<<t.next;
        }
        os<<endl;
    }
    return os;
}

template<int MAXS>
struct HashDFA{
    typedef NFA<MAXS> TNFA;
    typedef typename TNFA::SSet SSet;
    static const LL HX = 433494437;
    map<LL, SSet> hashToS;
    vector< map<LL, LL> > trans;

    const TNFA *pnfa;
    LL init(TNFA *nfa) {
        hashToS.clear(), trans.clear();
        assert(nfa);
        pnfa = nfa;
        trans.resize(nfa->MAXA);
        const VI& v = nfa->ss;
        SSet s0;
        _for(i, 0, v.size()) s0.set(v[i]);
        LL hash = calcHash(s0);
        hashToS[hash] = s0;
        return hash;
    }

    inline LL calcHash(const SSet& s){
        LL ans = 0, X = 1;
        _for(i, 0, MAXS) if(s[i]) ans += i * X, X *= HX;
        return ans;
    }

    inline bool contains(LL hash, int s) {
        assert(hashToS.count(hash));
        return hashToS[hash].test(s);
```

```
    }

    inline LL delta(LL s, int ch) {
        auto& m = trans[ch];
        if(m.count(s)) return m[s];
        const SSet& v = hashToS[s];
        SSet next;
        pnfa->delta(ch, v, next);
        LL hash = calcHash(next);
        hashToS[hash] = next;
        return m[s] = hash;
    }
};

const int MAXR = 500+4, MAXS = MAXR*4;
int main() {
    int n;
    string rex, txt;
    RexParser parser;
    NFA<MAXS> nfa;
    HashDFA<MAXS> hDfa;
    while(cin>>n>>rex>>txt) {
        nfa.init(rex, n+'1');
        _for(i, 0, n) nfa.add(0, 0, i+'0');
        nfa.remove_epsilon();
        VI ans;
        LL hs = hDfa.init(&nfa);
        _for(i, 0, txt.size()){
            if(i && hDfa.contains(hs, nfa.n-1)) ans.push_back(i);
            hs = hDfa.delta(hs, txt[i]);
        }
        if(hDfa.contains(hs, nfa.n-1)) ans.push_back(txt.size());
        cout<<ans<<endl;
    }
    return 0;
}
```

第 4 章　比赛真题选译

ACM/ICPC North America - Greater NY

1 循环最大值（Maximum in the Cycle of 1, North America - Greater NY 2011, LA5807）

如果 P 是整数序列 1～n 的一个排列，定义 P 中 1 循环最大值为 P(1), P(P(1)), P(P(P(1)))…的最大值。例如，假如 P 是如下的排列：

|1 2 3 4 5 6 7 8|

|3 2 5 4 1 7 8 6|

那么有：P(1) = 3，P(P(1)) = P(3) = 5，P(P(P(1))) = P(5) = 1。所以 1 循环最大值为 5。

对于给定的 n（$1 \leqslant n \leqslant 20$）和 k（$1 \leqslant k \leqslant n$），输出 n 的所有排列中 1 循环最大值为 k 的排列的个数。使用双精度浮点数 double 输出结果。

国王的高高低低（The King's Ups and Downs, North America - Greater NY 2012, LA6177）

国王有许多卫兵，希望把它们排成这样的顺序：每一个卫兵比身边的两个人都高或者都低，这样形成一条高低交错的线。例如，7 个卫兵身高分别是 160、162、164、166、168、170、172，那么就有如图 4.1 和图 4.2 所示的两种排序方式。

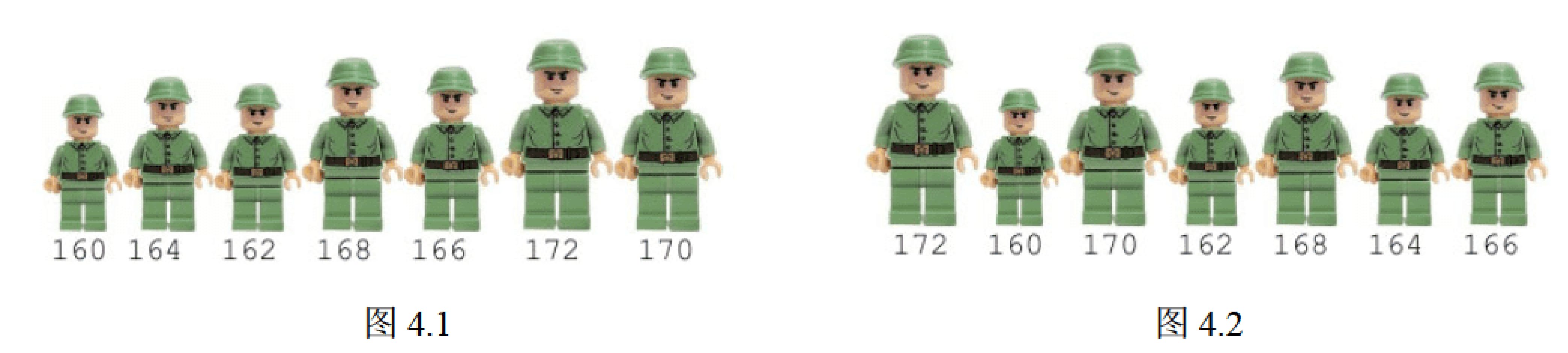


图 4.1　　　　图 4.2

给出身高都不同的卫兵的个数 n（$1 \leqslant n \leqslant 20$），计算有几种高低交错的排序方式，例如有 4 个卫兵 1、2、3、4，排序方式就有 1324、2143、3142、2314、3412、4231、4132、2413、3241 和 1423。

疯狂的兽医（Mad Veterinarian, North America - Greater NY 2012, LA6178）

有个疯狂的兽医开发了一个机器，可以把 1 种动物转换成其他的 1 种或者多种动物，并且可以反向转换。例如，有 3 个机器 A、B、C：

- ❑ A 可以把 1 个蚂蚁（a）变成 1 个海狸（b）。
- ❑ B 可以把 1 个海狸（b）变成 1 个蚂蚁（a）、1 个海狸（b）和 1 个美洲狮（c）。
- ❑ C 可以把 1 个美洲狮（c）变成 1 个蚂蚁（a）和 1 个海狸（b）。

我们能把一个海狸和一个美洲狮变成 3 个蚂蚁吗？是的：{b,c} C→{a,2b} −A 反向→{2a,b} A 反向→3a。能把 1 个蚂蚁变成 2 个蚂蚁吗？否。

转换规则如下：

（1）在正向模式下，每个机器都可以把 1 种动物变成有限非空的一个动物集合，动物的种类都是固定的。

（2）每个机器都可反向操作。

（3）每一种动物都有一个机器在正向模式下把它作为输入。

给出初始和目标的动物集合以及其中每种动物的数量，计算能否把初始集合变成目标集合，正向和反向转换都可用。并且需要找出最短的转换路径，本题中只有 3 种机器 A、B、C。

巧克力游戏（Chomp ACM/ ICPC North America - Greater NY 2013, LA6470）

Chomp 是一种二人制策略游戏，初始局面是一个由小方格组成的矩形巧克力块。玩家轮流选择一个格子，吃掉它以及所有在它右方和上方的格子。左下角的格子是有毒的，谁最后被迫吃掉这个格子就输了。图 4.3 就给出了一个以 3×3 大小巧克力为开始局面的游戏，其中×标记了有毒的格子。

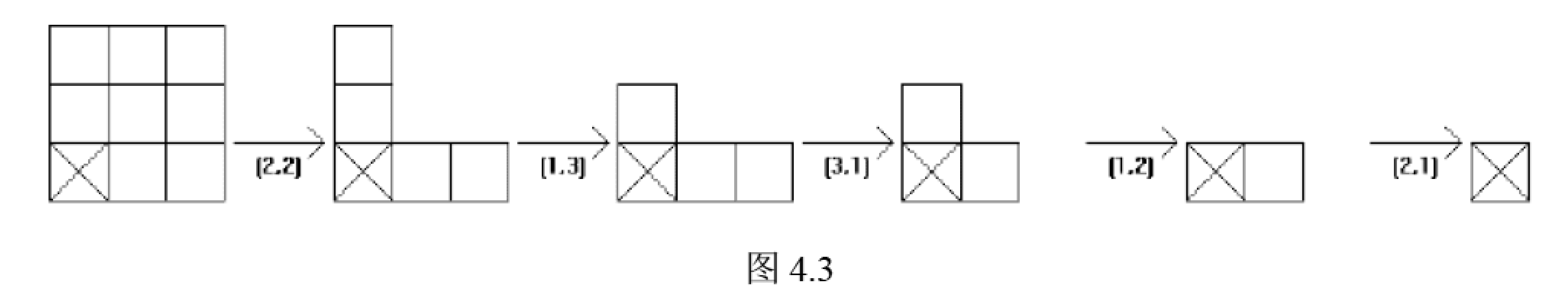


图 4.3

对于一个局面来说，如果有一种走法可以让对方必输，这个局面就是必胜局面。如果每一种走法都会留给对方一个必胜局面，那么这个局面就是必输的。对于一个 3×100 的 Chomp，输入多个局面，对于每个局面，确定它是必胜局面还是必输局面？如果是必胜局面给必胜走法。

ACM/ICPC Africa/Middle East - Arab

作家俱乐部（The Writer's Club, Africa/Middle East - Arab and North Africa 2007, LA4091）

一个网站上有许多作家，每个作家都被许多读者所喜欢。如果一个读者喜欢一个作家，他也有可能同时喜欢这个作家喜欢的其他作家的作品。例如，如果作家 John 喜欢 Alice 写的书，那么喜欢 John 的读者也有可能喜欢 Alice 的书。

进一步来说，网站希望给喜欢 John 的读者推荐 Alice 以及 Alice 喜欢的作家及 Alice 喜欢的作家喜欢的作家，如此等等。当然不能给读者推荐已经喜欢的作家。

输入 T（T<100000）个读者以及 N 个作家（N≤100），以及喜欢每个作家的人的姓名。根据这些数据，计算出需要将每个作家分别推荐给哪些读者，输出这些读者的姓名。

我想我会给自己买个足球队（Think I'll Buy Me a Football Team, Africa/Middle East - Arab and North Africa 2008, LA4367）

银行之间会有一些循环债务，如图 4.4（a）就显示了 4 个银行 A～D 之间的循环债务，



A 欠 B50，B 欠 A150 等，总共需要 380 来还清银行间所有的债务。

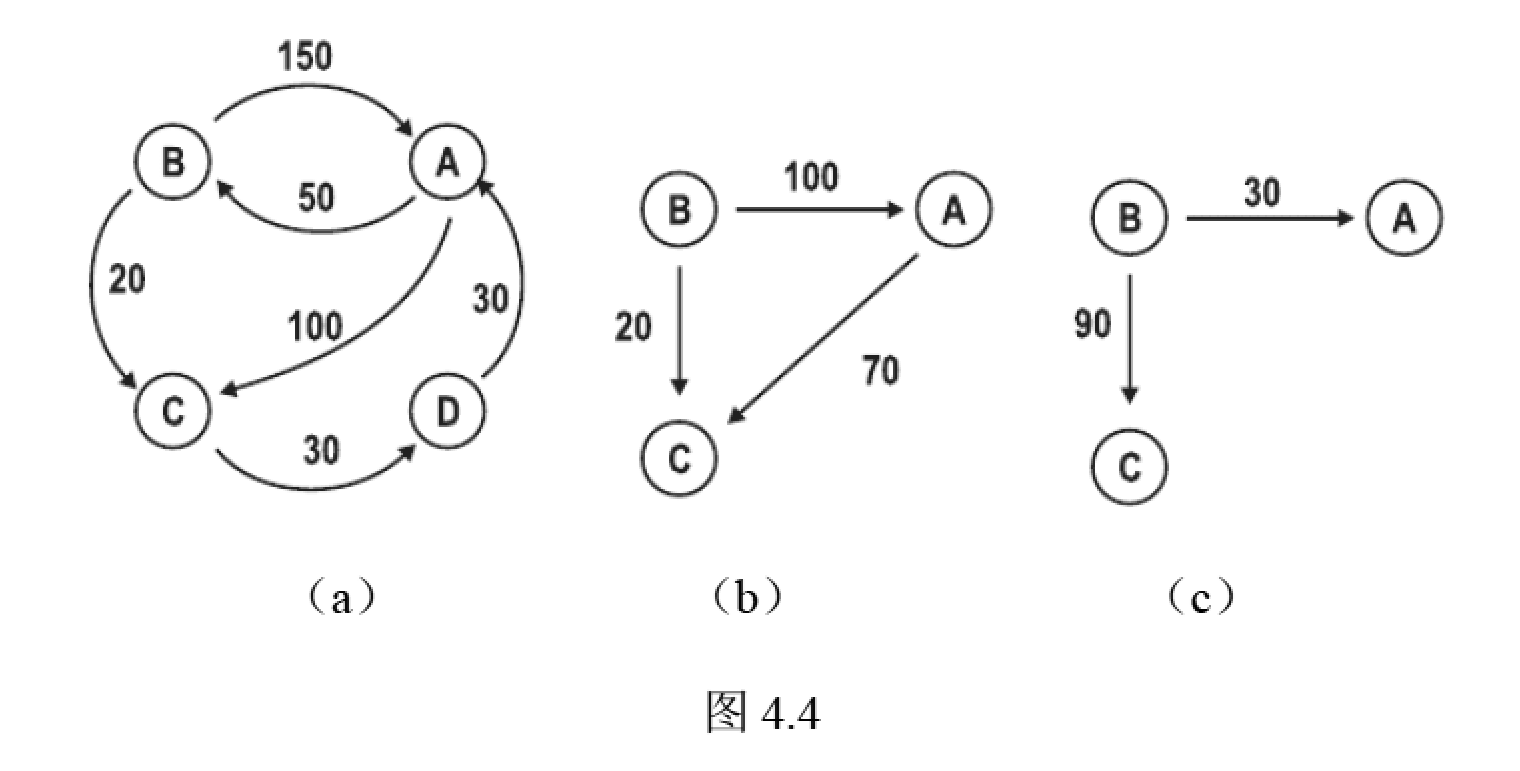


图 4.4

但是仔细观察可以发现，380 里面有许多实际上是被浪费的，可以采用如下策略优化：

（1）C 欠 D 和 D 欠 A 的一样，等于说 C 欠 A 是 30，把 *D* 排除出去。

（2）但是 A 欠 C 100，所以可以说 A 欠 C 70。

（3）AB 之间可以简化成 B 欠 *A* 100。这样就把图 4.4（a）简化成图 4.4（b），总共需要 190（减少了 190，也就是 50%）。

（4）可以继续优化，B 还 100 给 A，A 还 70 给 C，B 可以直接还 70 给 C（不用还 100 而是 30 给 A）。这样 120 就可以把所有债务还清，共节省了 260 也就是 68%。

给出 N（N<1000）个银行中任意两者的债务关系。输出在优化前后还清所有债务各自需要多少钱。

股市追捕（Stock Chase, Africa/Middle East - Africa and Arab 2009, LA4739）

股票市场需要禁止那种导致一个公司直接或者间接的控股自己的购买行为。例如，A 公司购买了 B 公司的股票，B 购买 C，C 再买了 A。前面两个合法。但是第 3 个就应该被拒绝，因为这样会导致 3 家公司间接对自身控股。给出按照时间顺序排序的购买交易，你的程序需要一次读入并且拒绝上述非法交易，其他的交易都要接受。

给出公司的个数 N（$0<N\leqslant 234$）以及 T（$0<T\leqslant 100000$）个交易：每个交易给出整数 A、B（$0<A,B\leqslant N$），表示 A 请求购买 B 的股票。输出要被拒绝的交易个数。

加密的密码（Encrypted Password, Africa/Middle East - Arab Contest 2012, LA6320）

有一种密码加密算法，它的输入是由英文小写字母组成的密码字符串，加密步骤如下：

（1）交换两个不同的字符的位置（可以做 0 到多次）。

（2）在第 1 步输出结果的前面附加 0 到多个小写英文字母。

（3）在第 2 步输出结果的后面附加 0 到多个小写英文字母。

第 3 步的结果就是加密结果。现在给出上述算法加密后的结果以及原始密码，计算这个加密结果是否可能是原始密码加密出来的。输入字符串长度都不超过 100000，由小写英文字母组成，并且原始密码的长度小于等于加密后的结果长度。

ACM/ICPC North America - Mid-Atlantic USA

化学品安全（Safety in Alchemy, North America - Mid-Atlantic USA 2007, LA3923）

一个罐子中装有多种化学品，两两起反应造成温度的升高。以如下的形式输入最多 64 对化学品以及其反应造成的温度升高值：

chemical1 chemical2 heat

表示 1 克的 chemical1 和 1 克的 chemical2 在一起会造成罐子升高 heat（0≤heat≤100）度。化学品的名称都是小写字母组成的长度 1～20 的字符串。

然后给出每种化学品的数量（在 0～1000 克之间）。计算罐子的温度最高可能升高多少度。

零件测试（Component Testing, North America - Mid-Atlantic USA 2012, LA6193）

为 n 类（$1 \leq n \leq 10000$）零件，每一类中的零件需要相同数量的不同检查者，但是不同类的零件需要的检查者数量可能不同，每一类给出零件数量 j（$1 \leq j \leq 100000$）和每个需要的不同的检查者个数 c（$1 \leq c \leq 100000$）。有 m 类员工（$1 \leq m \leq 10000$）。每一类给出员工的个数 k（$1 \leq k \leq 100000$）以及每个员工能被分配的零件个数 d（$1 \leq d \leq 100000$）。

每个员工可以检查任意一个可以是来自不同分类的零件，但是不能重复检查同一零件。计算检查这些零件，这些员工的数量是否足够。

卫星信号（Ping! North America - Mid-Atlantic USA 2013, LA6484）

你正在跟踪一些卫星，每个都会以固定的间隔发出 Ping 信号，每种信号的信号间隔都是唯一的。但是 Ping 信号会互相抵消：如果在一个时间点同时收到偶数个信号，那么你什么也听不到。如果是奇数个，你会收到一个 Ping 信号。在第 0 时间点，所有卫星都会发信号，之后以各自的间隔来发送。

给出一个长度在[2,1000]区间内的 Ping 信号序列，从中确定能听到的那些卫星的信号间隔。给出的信号序列，有可能不够长，导致某些卫星除了 0 时间点之外收不到第二个信号。这些卫星的信号间隔不需要计算。

稳定关系（A Stable Relationship, North America - Mid-Atlantic USA 2014, LA7118）

3D 打印机从顶端到低端一层层的布置原料来堆出一个物体。每一层都是要堆积体积和质量相同的打印原材料方块，每个都对应 3D 网格中的一个格子。同时假设打印机在打印过程中给物体施加的压力可以忽略。

在打印过程中，记当前底面为 P，已打印部分的质心在 P 上的投影为 C。物体不会倾倒的充要条件是，C 必须被一个 3 个顶点都位于已打印物体的最底面的形状内的三角形包含。3 个顶点可能被底面不同的区域所包含。如果在一层打印完了之后物体不会倾倒，那么这一层打印的过程中也不会倾倒。

给出物体的 3D 形状：宽度 w，长度 l，高度 h（$l,w,h \leq 200$），以及每一层的平面形状：



用 l 行 w 列的字符方阵来表示，“.”表示空格子，“#”表示实心。计算这个物体在打印完成之前是否会倾倒，只考虑物体在打印过程中和结束时是否会倾倒。

难对付的骑士（Tight Knight, North America - Mid-Atlantic USA 2014, LA7122）

骑士在棋盘上的每步移动可以是两种情况：

（1）水平两步加上垂直一步。

（2）水平一步加上垂直两步。

只要目标位置没被占用，就可以移动，不管中间经过的位置是否被占用。

给出一个 $n×m$（$1≤n≤1000$，$1≤m≤1000$）的棋盘。骑士一开始在(i,j)，行列都从 1 开始编号。有 c（$0≤c≤5000$）个障碍物。骑士不能落到障碍物上。

计算能不能最多增加一个障碍物就阻止骑士到达位置(k,l)（$1≤k≤n$，$1≤l≤m$）。(i,j)，(k,l)一开始都没有障碍物，并且这两个位置也不能放障碍物。

ACM/ICPC North America - Rocky Mountain

食物兑换券（Fast Food Prizes, North America - Rocky Mountain 2013, LA6444）

有一个餐厅在某些食物吃完之后送你一些贴纸。一些特定的不同贴纸的集合可用来换抵用券。如果一种抵用券需要的贴纸是 T1,T2,…,Tk，那么有这些贴纸各一张，就可以换到这种抵用券。每张贴纸只能用来换一种抵用券。当然，如果同一种贴纸如果你有多张，就可以各自用来换不同的抵用券。也有一些贴纸无法用来兑换。

输入 n（$1≤n≤10$）种抵用券的信息，以及编号为 1～m 的贴纸种类个数 m（$1≤m≤30$）。每种抵用券给出其价格（不大于 1000000），以及所需的 k（$1≤k≤m$）种贴纸编号。然后给出 m 个非负整数，其中第 i 个表示现在拥有的编号为 i 的贴纸个数（≤100）。计算用这些贴纸可以兑换到的抵用券的总价值。

表格（Tables, North America - Rocky Mountain 2013, LA6451）

HTML 用一种简单的标签格式来描述表格。需要用它来创建纯文字的 ASCII 艺术表格。表格可以认为是一个 m 行 n 列（$1≤m,n≤9$）的网格，每一个都是 2 字符宽 1 字符高的格子，例如一个 2*3 的网格的 ASCII 文字表示形式如图 4.5 所示。

```
 -- -- --
|11|12|13|
 -- -- --
|21|22|23|
 -- -- --
```

图 4.5

输出包含 $2m$*1 行，每一行 $3n$+1 个字符（奇数行包含首尾空格）。

但有些表格不是严格的基于网格的，因为某些格子可能要跨越多行多列。图 4.6 中，格 11 跨越 2 行，格 22 跨越 2 列。

```
 -- -- --
|11|12|13|
   -- --
|  |22   |
 -- -- --
```

图 4.6

给出一个 HTML 表格每一行中每一个格子占用的行列信息，输出这个表格的 ASCII-艺术纯文字形式。

化学品监测（Chemicals Monitoring, North America - Rocky Mountain 2013, LA6453）

一个负责环境数据检测的多 CPU 集群，所有 CPU 连接同一个输出生成单元（OGU）。每个 CPU 在处理完输入流之后都会立刻把结果送给 OGU，OGU 在瞬间完成输出处理。如果 CPU 处理完流之后，OGU 此时被占用，那么结果被丢弃。

每个输入流的时间段都是左闭右开区间$[s,s+d)$，其中 s 和 d 都是整数（$1\leqslant s,d\leqslant 10^9$），还有一个优先级 p（$0\leqslant p\leqslant 100000$）。CPU 的个数永远多于同一时间的数据流个数。根据输入流的优先级制定了如下规则：

（1）一个流到达时，系统可以选择接受或者拒绝。

（2）如果接受了，那么分配给这个流的 CPU 的唯一编号就压栈。

（3）只有编号在栈顶的 CPU 可以使用 OGU 来处理输出数据。使用完之后，此 CPU 编号出栈。

（4）如果多个流同时到达，那么对应的 CPU 编号可以以任意顺序压栈。

输入所有流的个数 n（$1\leqslant n\leqslant 5000$）以及各自的时间区间和优先级。需要在这些流中选择一个子集来处理，使得其中最终被 OGU 处理的流优先级总和最大化。

机票定价（Plane Ticket Pricing, North America - Rocky Mountain 2014, LA6867）

要对机票进行每周一次的定价。给出当前距离飞机起飞还剩下的周数 W，以及剩余座位数 N（$0<N\leqslant 300$，$0\leqslant W\leqslant 52$）。然后给出距离飞机起飞倒数第 i 周时，K（$0<K\leqslant 100$）种不同的票价 $p_1\sim p_k$($0<p_1<\cdots<p_k<1000$)，以及每种票价对应的能够卖出去的票数 $s_1\sim s_k$（$0\leqslant s_i\leqslant N$）。

计算第 W 周该如何定价才能保证航空公司在起飞之前获得最大的收入。输出获得最大收入的票价，如果这样的票价有多个，输出最小的那个。

餐厅评级（Restaurant Ratings, North America - Rocky Mountain 2014, LA6872）

旅行网站设计了一个餐馆评级系统，每个餐馆都由 n（$1\leqslant n\leqslant 15$）个评论家来打分，每人打一个正整数的分数（越高越好）。餐馆的排名规则是先按照各个评论家的打分总分（不超过 30）排序。如果总分相同，就按照 1～n 这 n 个评论家的 n 个打分的字典序排序。

现在给出一个餐馆的得分，计算按照以上排名规则，排名不超过这个得分的所有可能的打分结果的个数。输出保证可用 64 位有符号整数存放。

锁着的宝藏（Locked Treasure, North America - Rocky Mountain 2014, LA6873）

有 n（$1\leqslant n\leqslant 30$）个强盗把宝藏锁在一屋内，必须至少有 m（$1\leqslant m\leqslant n$）个一致同意才

能去取宝藏。他们在门上放很多锁，必须同时打开才能开门。每个锁可以配不超过 n 把钥匙，分别发给一些强盗。一组强盗当且仅当其中有人有这把锁的钥匙时才能打开这把锁。

给出 n 和 m，计算最少需要多少把锁才能保证：在钥匙分配合理的前提下，任何组强盗只有在人数不少于 m 的情况下才能打开锁宝藏的门。

举例来说，如果 n=3，m=2，需要 3 把锁就行了。锁 1 的钥匙给强盗 1 和 2，锁 2 的钥匙给强盗 1 和 3，锁 3 的钥匙给强盗 2 和 3。没有一个强盗能独自打开锁，但是任何两个强盗组成一组就可以打开所有的锁。需要思考一下为什么 2 把锁不能满足条件。

连分数（Continued Fraction, North America - Rocky Mountain 2014, LA6875）

一个实数 r 的简单连分数表示是一个迭代的，把 r 写成一个整数和另外一个数的倒数之和的过程。表示形式如下：

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}$$

其中，a_i 都是正整数。我们称其为部分商。若 r 是有理数，则 a_i 的数目有限。

现在给出两个有理数 $r1$ 和 $r2$（$r1$>$r2$>0）的连分数表示形式，对这两个有理数进行基本的四则运算，并且输出结果的连分数形式。$r1$ 和 $r2$ 部分商的个数 i 满足 1≤i≤9，且 a_i≤10。

ACM/ICPC North America - East Central NA

人名追踪（What's In A Name?, North America - East Central NA 2001, LA2354）

FBI 正在监控一个犯罪窝点，里面有 n（n≤20）个嫌疑人，都有唯一 ID。FBI 记录了一系列按照时间顺序排列的人员进出（使用人名）的情况，以及窝点向外发送消息的记录（使用 ID）。所有的 ID 以及人名都会在记录中出现，一开始窝点是空的。所有的人名和 ID 都只包含最多 20 个小写字母。

根据这些记录计算出 ID 和人名的对应关系，按照人名的字典序输出。如果根据记录无法确定一个人名对应的 ID，就输出“???”作为 ID。

字母排序（Sorting It All Out, North America - East Central NA 2001, LA2355）

对于前 n 个大写字母（2≤n≤26），输入 m 个形如 A<B 的关系，表示字母 A 排在 B 前面。根据输入顺序对 n 个大写字母按照上文给定的顺序排序并且输出结果。如果无法确定顺序或者给定的 m 个关系互相矛盾，也输出相应的结果（具体输出格式请参考原文）。

二叉树排序（Trees Made to Order, North America - East Central NA 2001, LA2357）

对于所有二叉树，按照如下规则进行编号：

- 空树编号为 0。
- 只有一个结点的树编号为 1。
- 所有 m 个结点的树的编号比 m+1 结点的都小。

- ❑ 对于编号为 n 的 m 个结点的树，左右子树分别是 L 和 R。所有 m 个结点且编号大于 n 的树必须满足：左子树编号大于 L 的编号或者左子树为 L 且右子树编号必须大于 R。

按照以上规则，编号前 10，以及 w 为 20 的树如图 4.7 所示。

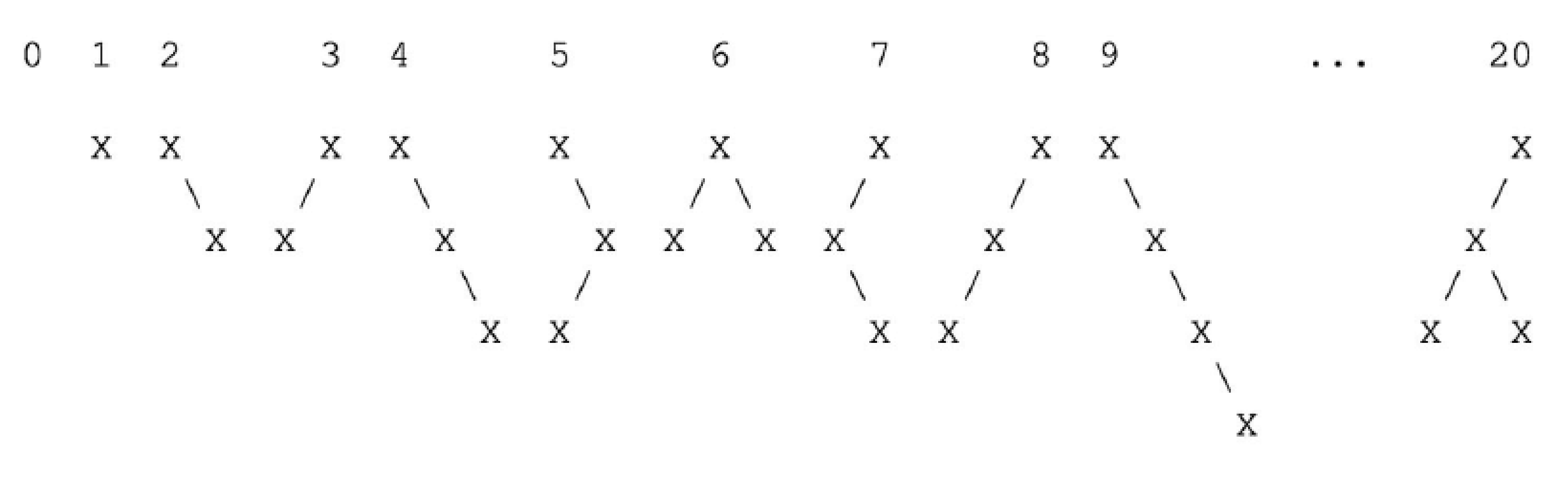


图 4.7

给出编号 n（$1 \leqslant n \leqslant 500000000$），输出 n 对应的树。输出格式如下：

- ❑ 无子结点的树，输出 X。
- ❑ 左右子树分别为 L,R 的树输出为(L')X(R')，L'和 R'分别为 L 和 R 的表示。
  - ➢ 如果 L 为空，输出 X(R')。
  - ➢ 如果 R 为空，输出(L')X。

空间站防御（Space Station Shielding, North America - East Central NA 2001, LA2358）

火星低轨道的空间站由一系列的立方体单元组成，在穿过火星大气顶层时有可能遇到一些致命的细菌，要在所有靠外的表面加装防护。可认为方块的面-面接触和边-边接触是足够密封的，细菌无法进入。

空间站可以装在一个 $n \times m \times k$（$1 \leqslant n,m,k \leqslant 60$）的网格立方体中。每个单元格都可能有一个空间站的单元。如图 4.8 所示，所有的单元格编号为 0, 1, 2,⋯, $k*m*n-1$。

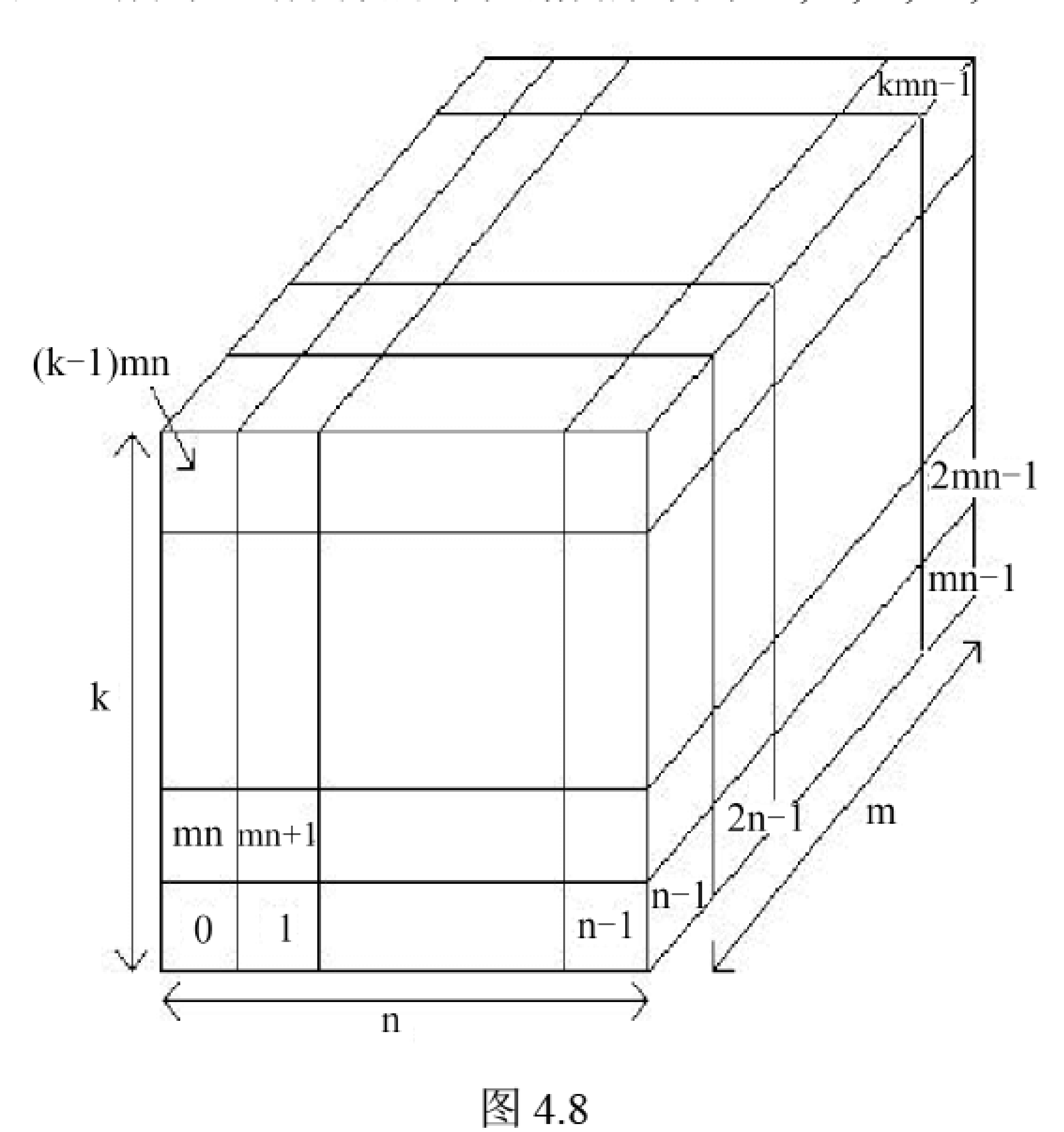


图 4.8



空间站包含 L 个单元，输入每个单元的编号（每一行 10 个）。输出暴露在最外面的立方体表面个数。

机器人（Robots, North America - East Central NA 2001, LA2360）

有一种在 31×31 棋盘上进行的单人游戏。每个格子使用坐标（r,c 从 1 开始）来表示。格子可能是空的，可能是你占的，可能被机器人占的，也可能是残骸。游戏的目标是不停地移动，在被机器人杀掉之前杀掉所有的机器人。

一开始你位于(15,15)，并且有 R（$1\leqslant R\leqslant 50$）个机器人在其他格子。剩余格子皆空。同时给出 T（$0\leqslant T\leqslant 20$）个潜在的传送位置。你先移动，然后就和机器人轮流移动。

每一步，你可以在 8 个方向中任选一个移动一格，或直接传送到指定的传送位置，或原地不动。如果要移动到另外一格，这个格子要么是空的，要么沿着行走方向把上面的残骸推到下一个不包含残骸的格子中去。如果这个格子上有机器人，机器人就被杀掉。如果是传送，传送的目标必须是个空格子。不能走出棋盘边界或者把残骸推出边界。

机器人移动时，每个机器人都在 8 个方向的邻居（即使不是空格）中选择距离你当前位置最近的那个移动过去。两格子（r_1,c_1）和（r_2,c_2）之间的距离定义为$|r_1-r_2|+|c_1-c_2|$。每次移动所有的机器人都走一步。如果多个机器人走到同一个格子，或者机器人走到残骸的格子，这些机器人就被杀掉变成残骸。

如果有机器人走到你的当前位置（即使是多个机器人同时走过来变成残骸），你就输了。如果所有机器人都被杀掉并且你没被碰到，你就赢了。

为了在游戏中停留尽量长的时间，你只选择不会导致立刻被杀掉的移动方式。一个貌似合理的策略是，走到一个格子或者原地不动，使得在你移动之后接着机器人再移动后的机器人数量最小化。如果有多重方案，选择可以让你下一次移动之前机器人离你距离的最小值最大的那个。如果还有多重方案，依次选择目标行和列最小的那个。

如果无论如何移动都会导致立刻被杀掉，只要可以不被立刻杀掉，就传送到预先指定的位置之一。选择传送位置时，要在给定列表中从前往后搜索。如果没有合适的传送位置可以让你存活，那就选择原地不动，接着输掉。

给出机器人的初始位置。首先输出游戏过程中传送过的位置，然后输出游戏的结果。需要注意的是，本题输出格式较为复杂，详情请参考原题。

道路规划（Roads Scholar, North America - East Central NA 2001, LA2359）

高速公路的路网可以进行如下定义：有编号为 $0,1,\cdots,n-1$ 的 n（$5\leqslant n\leqslant 30$）结点（其中有 k 个结点刚好也是城市），连接这些结点的是 m 条路，对于第 i 条路输入 $i1,i2,d$，表示它是连接结点 $i1$ 和 $i2$ 长度为 d 的双向道路。

对于每个城市输入其结点编号和名称（最多 18 个字符）。需要在路网上安装 s 个路牌。其中对于第 i 个路牌输入 $i1,i2,d$，表示要在结点 $i1$ 到 $i2$ 的路上且距离 $i1$ 为 d 的地方安装一个路牌。路牌上要标明距离城市 X 的距离，X 必须满足：$i1$ 到 X 的最短路径必须经过 $i1\rightarrow i2$ 这条路。

计算并输出每个路牌上需要印上的城市名称以及相应的路牌到城市的距离，按照距离

从小到大排序。

需要注意的是，本题输出格式较为复杂，详情请参考原题。

抛球（Ball Toss, North America - East Central NA 2002, LA2581）

有编号为 1, 2, …, n 并且按照编号顺序以顺时针站成一圈的 n（$2 \leqslant n \leqslant 30$）个同学，每个人心中都有一个向左或者向右的想法。一开始球在 1 手中，并且把球扔到 k（$k>1$）手中。之后抛球规则如下：

（1）一个人在接到球之后，如果此时心中向左。他就把球扔到抛球者的从他看来的左方的同学手里，并且心中想法变成向右。

（2）一个人在接到球之后，如果此时心中向右。他就把球扔到抛球者的从他看来的右方的同学手里，并且心中想法变成向左。

（3）如果一个心中向左的人接到他紧左边的同学的来球，他就把球抛到他的紧右边同学那里，并且心中想法变成向右。

（4）如果一个心中向右的人接到他紧右边的同学的来球，他就把球抛到他的紧左边同学那里，并且心中想法变成向左。

（5）上述两条规则是为了避免有人接球之后把球抛给自己。不管一开始大家心里想的方向如何，和一开始谁先抛球，最终每个人都会有机会抛球。现在需要计算一下经过几次之后每个人都算抛过球了，并且输出最后一个抛球者的编号。

递增序列（Increasing Sequences, North America - East Central NA 2002, LA2583）

给出一个数字组成的字符串（长度≤80），插入一些逗号来把它分成一个递增的整数序列（整数可以包含前置 0），并且要求最后一个数字尽量小。输出这个序列（以逗号分隔）。如果存在多个可能的序列，输出其中字典序最大的。

前序后序（Pre-Post-erous!, North America - East Central NA 2002, LA2584）

一般来说，使用先序和后序遍历序列是无法唯一确定一棵二叉树的。对于 m 叉树也一样。给出一颗 m（$1 \leqslant m \leqslant 20$）叉树的先序和后序遍历序列 s1, s2，二者长度均为 k（$1 \leqslant k \leqslant 26$）。序列中会用到前 k 个小写字母。计算出可能有多少种树的先序和后序遍历序列是 s1 和 s2，输入保证结果可用 32 位有符号整数存储，并且至少有一棵树符合条件。

淘汰赛（Knockout Tournament, North America - East Central NA 2002, LA2585）

在一场编号为 $1,2,3,\cdots,2^n$ 的 2^n（$n<8$）个选手的淘汰赛中，选手输一场就会被淘汰掉。赢的人留下来继续结对比赛直到只剩下 1 个选手。第一轮的比赛就在 $2k-1$ 和 $2k$ 之间举行，$k = 1,2,\cdots,2^{n-1}$。那么比赛结果就形成了一个完全二叉树，如图 4.9 所示中 n=3。

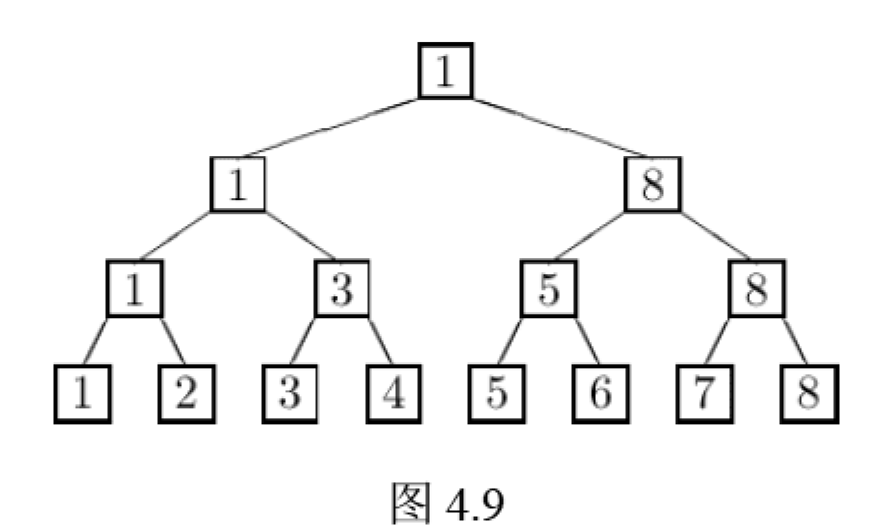
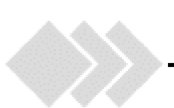


图 4.9



但是对于最终选手的排名却有争议。假设 A 赢 B，B 又赢 C，那么可以认为 A 也可赢 C，也就是说输赢是有传递性的。那么谁是第 1 名选手就很确定了。问题是一个选手可以声明的最高名次和最低排名。如在图 4.9 中，2 只是输给了 1（第 1 名），可以认为它最高可能是第 2 名，最低是第 8 名。5 可以声明的最高名次是第 3（输给了最高可以声明是第 2 名的 8），但是最低是第 7（在第一轮中赢了）。

给出比赛结果，计算指定的一个选手集合中每个人可能的最高和最低排名。

装饰（Decorations, North America - East Central NA 2003, LA2825）

有 n 种（$n \leqslant 26$）数目不限的珠子（种类用不同的大写字母表示），选择 l（$l \leqslant 100$）个连成一串。同时珠子的种类形成一个字符串。给出 m（$m \leqslant 600$）个长度均为 k（$1 \leqslant k \leqslant 10$）的字符串。计算有多少种选择 l 个珠子串起来的方案，使得形成的字符串的所有长度为 k 的子串是给出 m 个其中之一。输入保证结果可用 32 位整数存放。

EKG 序列（EKG Sequence, North America - East Central NA 2003, LA2826）

EKG 是一个正整数序列，按照以下规则生成：

- 前两项是 1,2。
- 后面每一项都是和前一项有公因子（1 除外）的最小的未在序列中出现过的整数。

所以第 3 项是 4（最小的未使用过的偶数），第 4 项是 6，第 5 项是 3。这个序列的前面部分项如下： 1, 2, 4, 6, 3, 9, 12, 8, 10, 5, 15, 18, 14, 7, 21, 24, 16, 20, 22, 11, 33, 27。 EKG 序列有两个特殊性质：

（1）所有的正整数都会出现在序列中。

（2）所有的素数都是以递增序出现在序列中。

对于一个给定的整数 n（$1 \leqslant n \leqslant 300000$），计算出它在序列中的位置（以 1 作为起始位置）。注意包含所有小于等于 300000 整数的 EKG 序列，不会包含任何大于 1000000 的整数。

四分树（Squadtrees, North America - East Central NA 2003, LA2830）

四分树可以用来存储黑白双色的简单位图（黑 1 白 0）。建树方式如下：

（1）建立一个根结点表示整个图像。

（2）如果图像全是 1 或 0，那么把颜色存储到根结点，结束。

（3）否则把图像分成 4 个区域，从左到右，从上到下依次把 4 个子块构造成 4 棵树作为根结点的子树。

图 4.10 中，左侧图像和右边的树对应。

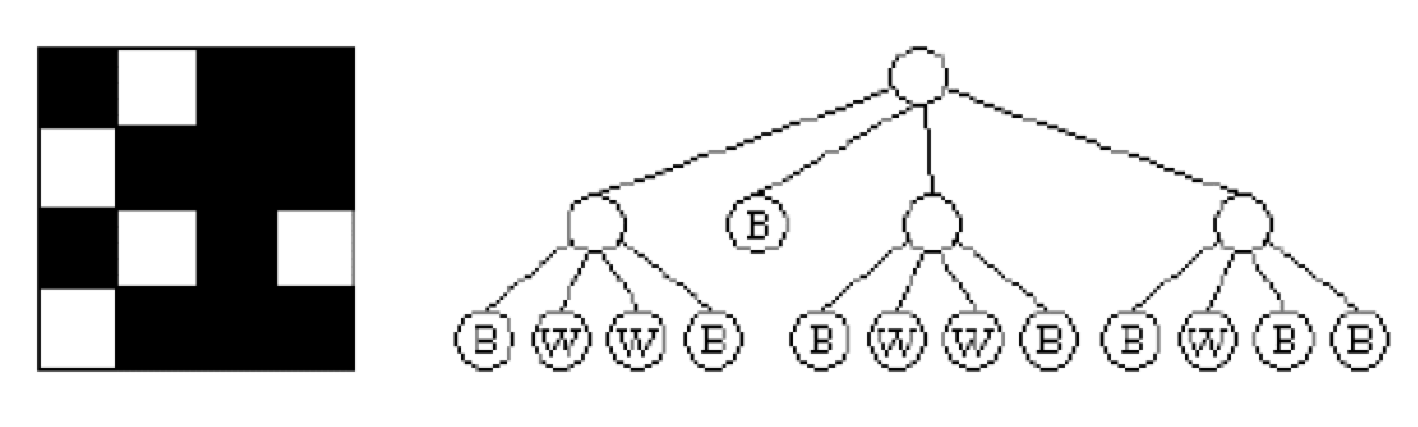


图 4.10

这个过程仅仅适用于边长为 2 的整数次方的正方形图像。如果不符合这个条件，在

图像的右边和下边补零，使之变成一个符合条件的图像。例如，5*13 的图像就会被补成 16*16 的。

如果在四分树中把重复的子树找出来，其所占空间可以进一步压缩。图 4.10 中的四分树就可以压缩成图 4.11 所示的结构。

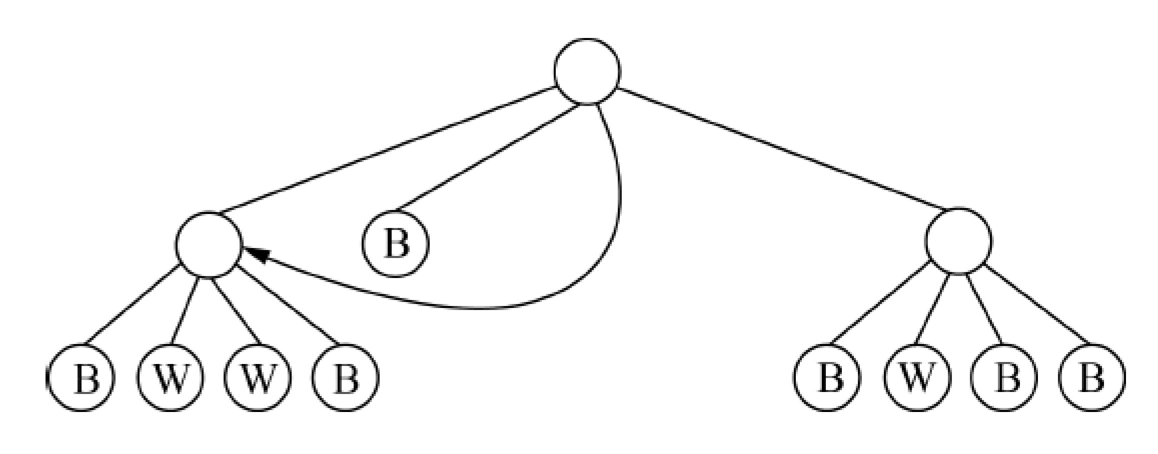


图 4.11

我们称这种结构为超级四分树，当然只有替换哪些高度大于 1 的树才有意义。根据这个规则，上述的超级四分树仅仅包含 12 个结点。

给出一个 $n*m$（$n,m \leqslant 128$）的图像，确定其四分树和超级四分树表示形式中的结点数目。

反素数序列（Anti-prime Sequences, North America - East Central NA 2004, LA3079）

给出连续整数序列 $n,n+1,n+2,\cdots,m$，其反素数序列是这些整数的符合如下条件的重排：每个相邻数之和都不是素数，如 n=1,m=10，其中一个反素数序列就是 1,3,5,4,2,6,9,7,8,10。它也是按照字典序最小的那个。进一步定义 d 阶反素数序列如下：其任意长度为 $2,3,\cdots,d$ 的连续子序列的和都不是素数。

上文中的序列就是 2 阶，但不是 3 阶，因为 5,4,2 的和为 11。按照字典序第一个 3 阶反素数序列就是 1,3,5,4,6,2,10,8,7,9。

输入 n,m,d（$1 \leqslant n < m \leqslant 1000$，$2 \leqslant d \leqslant 10$），输出字典序最小的那个的 d 阶反素数序列，以逗号分隔。如果不存在，直接输出“No anti-prime sequence exists.”。

挖沟（I Conduit!, North America - East Central NA 2004, LA3081）

输入 n（$n \leqslant 10000$）条长度非零的线段，每条线段都给出形如 x1 y1 x2 y2 的坐标，其中每一个坐标值都是[0,1000]区间内的小数点后最多 2 位的浮点数。

如果两条共线线段有交集，就可以合并成一条，两条以上线段也一样。计算 n 个线段合并完了之后的线段数目。

掷骰子游戏（Roll Playing Games, North America - East Central NA 2004, LA3082）

给出 n（$1 \leqslant n \leqslant 20$）个骰子，不一定都是 6 面的。输入每个骰子的面数 f（$3 \leqslant f \leqslant 20$）以及每个面上的数字（1～50 之间的整数）。现在需要计算最后一个骰子，它必须满足以下条件：

（1）面数必须是给定的 r（$4 \leqslant r \leqslant 6$）。

（2）给出 m（$1 \leqslant m \leqslant 10$）个数字，$v_1,\cdots,v_m$，以及对应的 $c_1,\cdots,c_m$。要求对于每个 i=1～m，包括这个骰子的 n+1 个骰子有 v_i 种方式掷出来之后的值之和为 c_i。其中 v_i 和 c_i 都严格小于 32 位有符号整数的最大值。



按照递增序输出所求筛子每个面的数字。如果问题有多解，输出字典序最小的那个。

翻译（Translations, North America - East Central NA 2004, LA3085）

有一个两种语言中短语的对应列表，每一行都包含两个短语。本来是意义对应的，但对应关系错乱了，形成如下样式的列表：

语言 1 的短语	语言 1 的短语
arlo zym	bus seat
flub pleve	bus stop
pleve dourm	hot seat
pleve zym	school bus

因为短语都是两个词的，所以可以还原出原来的对应关系，例如，从上表可还原出 arlo→hot, zym→seat, flub→school, pleve→bus, dourm→stop。

输入两种语言的短语（只包含两个单词）列表各 n（n≤250）个，单词都由字母组成。每个语言的短语列表中包含的不同单词数不超过 25 个，每一个单词在短语中作为第一个和最后一个单词出现的次数不超过 10。按照 word1/word2 的形式输出单词的对应关系，其中 word1 是语言 1 中的单词。按照 word1 字典序从小到大输出，且不允许出现重复。输入保证结果唯一。

Efil 的游戏（The Game of Efil, North America - East Central NA 2005, LA3374）

著名的 Game of Life（生命游戏）是在方格组成的矩形区域上进行，每个方格有上下左右 8 个方向的邻居，每个格子都可能有生命。繁殖后代的规则如下：

（1）如果被占用的格子包含 0,1,4,5,6,7,8 个被占用的邻居，生命死亡（0,1 死因：太孤独，4～8 死因：太拥挤）。

（2）如果被占用的格子有 2～3 个被占用邻居，生命存活。

（3）如果空格子有 3 个邻居有生命，新生命在此诞生。

（4）区域板的上下边是连通的，左右边也是连通的。例如，顶边的格子上方的邻居位于底边。

对于 m 行×n 列（m×n≤16）的一个区域以及其上生命的位置，输出它上一代可能有多少种布局？举例来说，对于图 4.12 所示的布局：

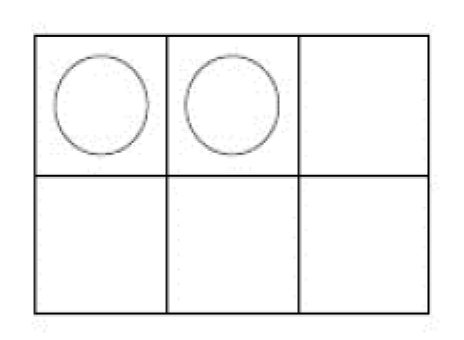


图 4.12

有 3 种可能的上一代布局，如图 4.13 所示。

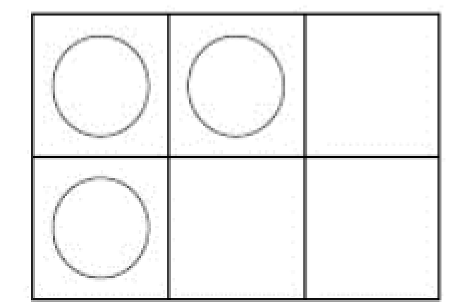
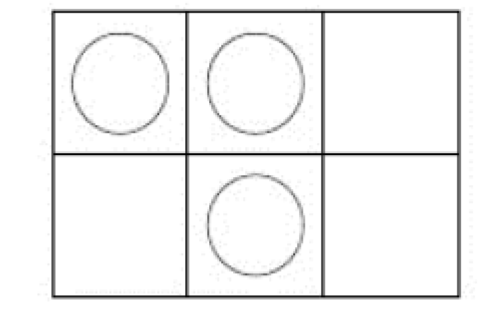
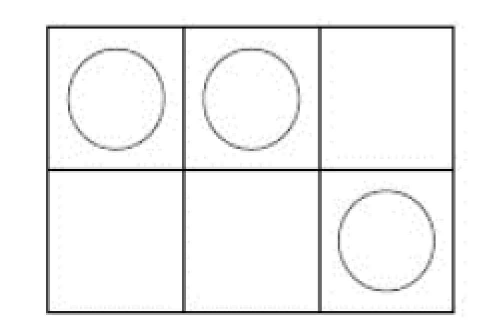
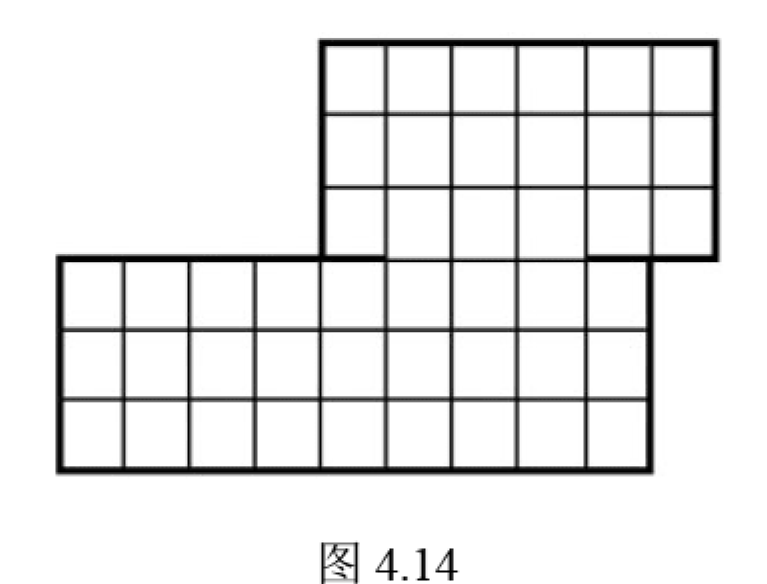




图 4.13

如果所求结果为 0，直接输出“Garden of Eden.”。

正方形计数（Square Count, North America - East Central NA 2005, LA3377）

博物馆包含 n（n≤1000）个房间。每个房间都是一些方格子组成的矩形，给出其一组对角顶点的整数坐标 $x1,y1,x2,y2$（坐标值≤1000000）。房间的区域都不会重叠，但是边和边可能有重叠部分。


图 4.14

需要计算所有房间组成的图案中总共有多少个正方形。图 4.14 中就包含 86 个，45 个 1×1，28 个 2×2，13 个 3×3，注意两房间之间的门长度只有 3，而正方形不能被墙隔开。如果两个房间的边公共长度为 m，那么门长度为 m−2，而且位于重叠部分的正中间。

输入 n 个房间的坐标，输出正方形的个数。结果保证能用 32 位整数存储。

两端游戏（Two Ends, North America - East Central NA 2005, LA3379）

在双人玩的“两端”游戏中，有 n（n 是偶数且 n≤1000）张卡片排成一行。每张卡片都朝上，并且写着一个正整数。玩家轮流从两端选择一张拿走后放到自己的卡片堆里。最后卡片堆中数字之和最大的那个玩家赢。一种贪心的选择策略是永远选择两端之中最大的那个卡片（如果相等就选左边的）。但是这种策略不一定是最优的，在下面的例子中（第一个玩家选择 3 而不是 4 就赢了）：

3 2 10 4

依次输入 n 张卡片上面的数字（数字之和≤1000000），假如玩家 1 自由选择最优策略，玩家 2 选择上述的贪心策略，玩家 1 先手。记最后玩家 1 卡片堆数字之和与玩家 2 的卡片堆数字之和的差为 P，计算 P 的最大值。

判断毛毛虫（Caterpillar, North America - East Central NA 2006, LA3724）

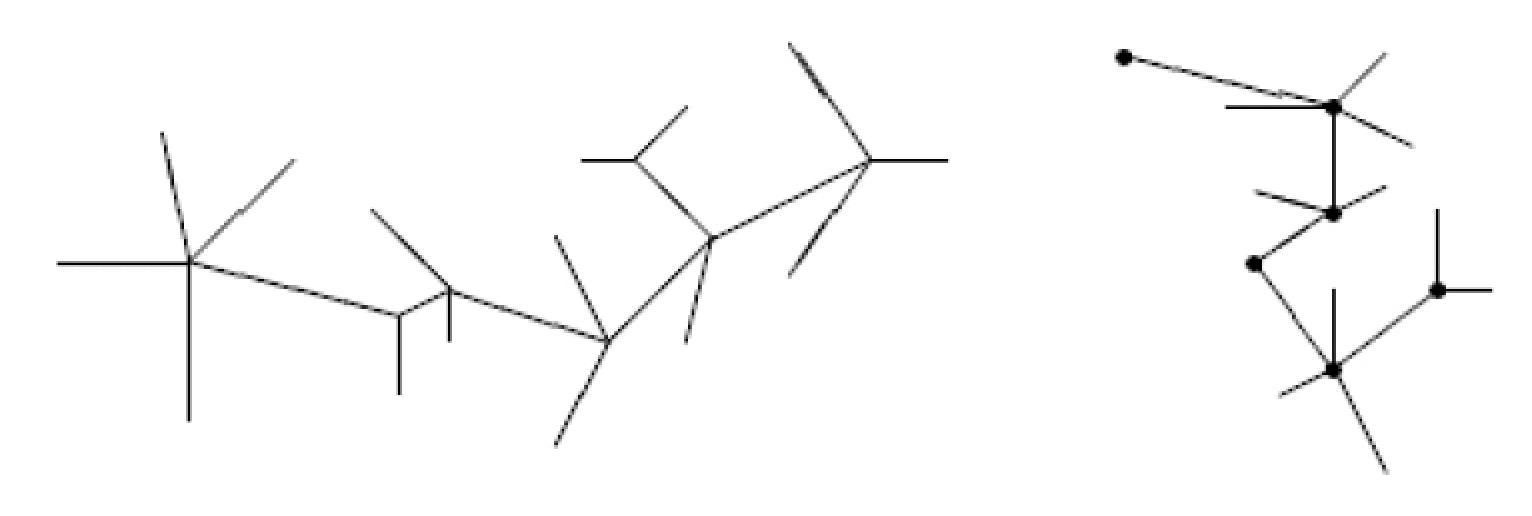
如果一个无向图满足以下条件，就称其为毛毛虫。

（1）联通。

（2）无环。

（3）存在一条路径使得每个结点在路径上或者有一邻居在路径上。

这条路径就称为毛毛虫的脊，可能脊并不是唯一的。如图 4.15 左图就不是毛毛虫，而右图就是，且其中的圆点所在的路径就是其中的一条脊。


图 4.15

给出编号 1～n 的 n（n≤100）个结点，以及连接这些结点的 e（e≤300）条边所组成的



图（未必是联通且无环）。判断这个图是否是毛毛虫。

饰板装箱（Plaque Pack, North America - East Central NA 2006, LA3728）

有 n（$1\leqslant n\leqslant 100$）个宽度都为 w（$1\leqslant w\leqslant 10$）的饰板需要装进高度为 b（$1\leqslant b\leqslant 100$）、宽度为 w 的箱子中。对于每个饰板，给出其高度 h（$1\leqslant h\leqslant 10$，$h\leqslant b$）。然后给出一个 $h\times w$ 的字符矩阵表示其形状，字符“X”表示饰板的部分，“.”表示空。饰板的形状各异，图 4.16 中有 3 个例子。

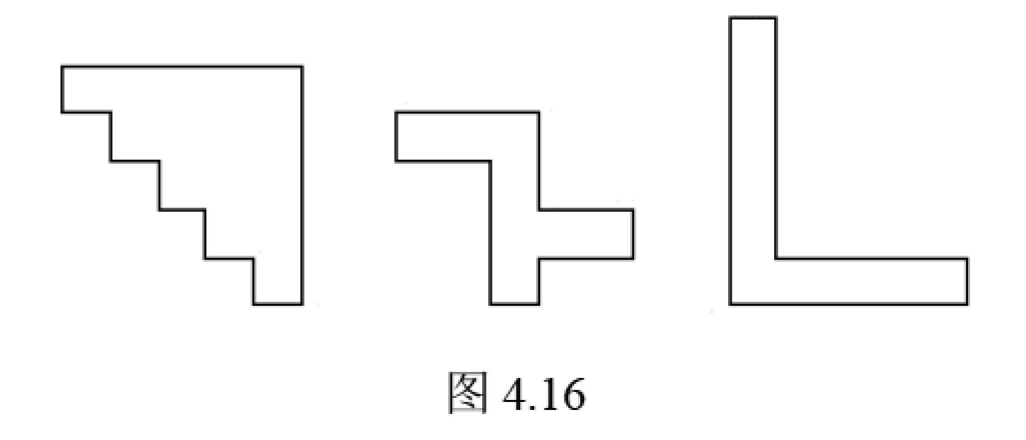


图 4.16

把饰板往箱子中装时，它会一直往下滑直到某些部分碰到盒子的底部或者其他饰板的最高点。如果从左到右往箱子中装饰板，会形成图 4.17 左边的布局。从右往左会形成图 4.17 右边的布局。

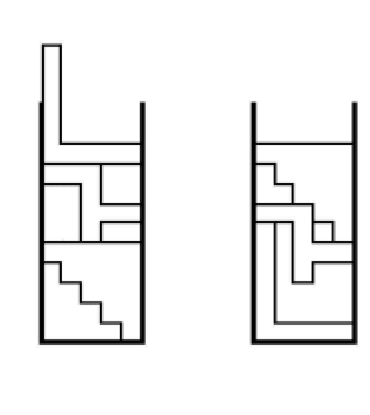


图 4.17

当一个新的饰板装不下时，先把盒子封起来，然后用一个新盒子装。对于输入的 n 个饰板，输出按照输入顺序安装所用到的每个盒子中饰板堆积的最大高度。

屋顶设计（Roofing It, North America - East Central NA 2006, LA3729）

给出一个房屋的一侧的屋顶形状，包含 n（$2\leqslant n\leqslant 100$）个点及其坐标$(x_i,y_i)$（$0.0\leqslant x_i, y_i\leqslant 10000.0$），按照 x 的递增序给出，并且保证最后一个点的 y 值最大，如图 4.18 所示。需要把设计修改成连起来的 k（$1\leqslant k<n$）条线段，满足以下条件：

（1）必须是上凸的。

（2）原设计中的 n 个点都不能在线段的上方。

（3）所有的（x_i,y_i）距离新线段距离的最大值要最小。

（4）原设计中至少有两个点刚好在线段上。

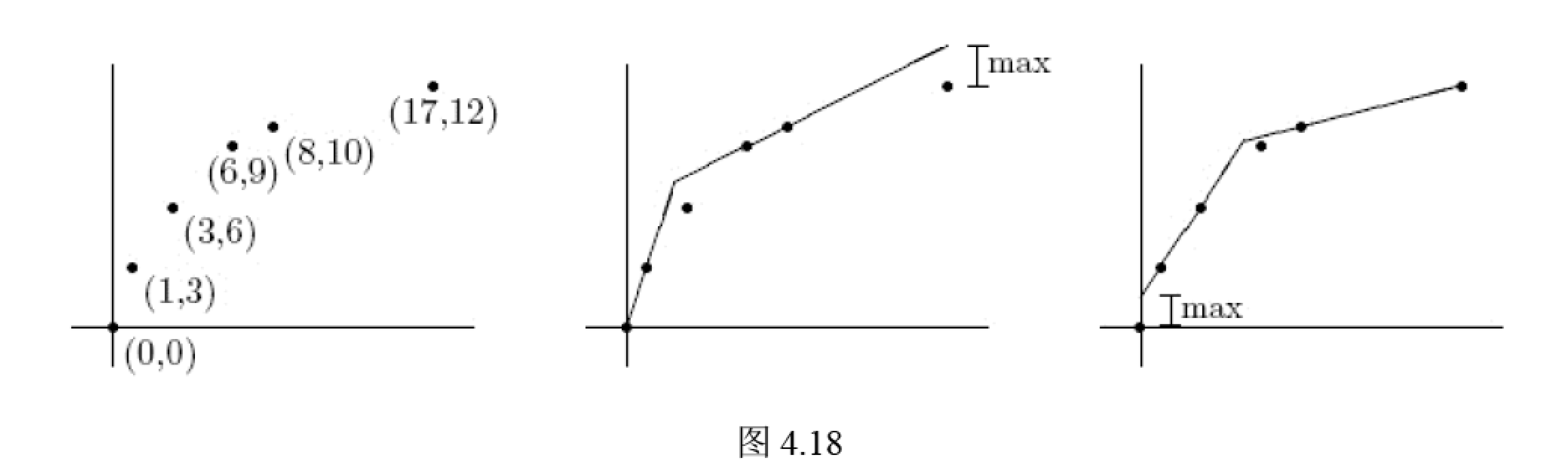


图 4.18

右边两个图都是凸的，但是最后一个就把点到线段的距离最大值最小化了。输出符合条件的设计中，每个点到新线段的距离的最大值，四舍五入保留小数点后 3 位。

罗马数字加法（CIVIC DILL MIX, North America - East Central NA 2007, LA3963）

罗马数字包含如表 4.1 所示的 7 种符号。

表 4.1

符号	I	V	X	L	C	D	M
值	1	5	10	50	100	500	1000

符号 I、X、C、M 可以按需重复（I、X、C 不能超过 3 次），所以 3 就表示为 III，27 是 XXVII，4865 是 MMMMDCCCLXV。这些符号都是递减序排列。但有一个例外：如果一个值更小的符号写在大的符号前面，它的值要从大的符号那里减掉。例如，4 就写成 IV 而不是 IIII，并且 900 要写成 CM。

这种减法的规则如下：

（1）只有 I、X、C 能用来减其后更大的符号。

（2）只能减一次（如 8 不能写成 IIX）。

（3）只能出现在值不超过它 10 倍的符号前面（所以 99 不能写成 IC 而要写成 XCIX，499 不能写成 XD 而要写成 CDXC）。本题标题中的前两个 CIVIC 和 DILL 就不是有效的罗马数字，而 MIX 有效。

输入 n 个罗马数字，计算它们的和（<5000）并输出其罗马数字表示形式。

吸引人的折纸（A Foldy but a Goody, North America - East Central NA 2007, LA3964）

有一个纸条。有两种命令对纸条进行折叠：U 表示纸条的右端被叠到左端的上方。L 表示右端被叠到左端的下方，如图 4.19 所示。



图 4.19

经过几次折叠之后，需要每次围绕折痕展开 90°。图 4.20 中就展示了经过一次 U 和一次 L 折叠之后再展开的情形。

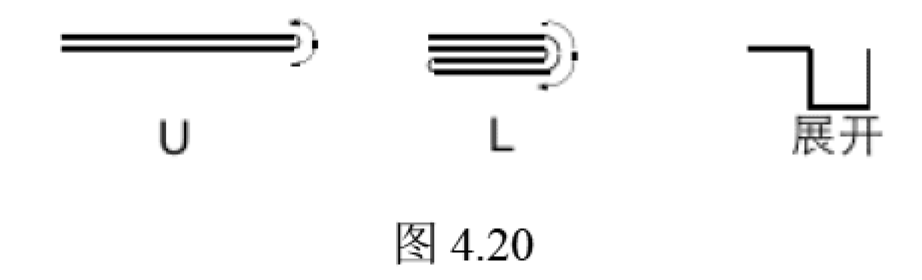


图 4.20

定义纸条的左端坐标是(0,0)，第一个直角的坐标是(1,0)。如果折叠了 n 次，第 0 个点为左端点，展开之后的右端点就是第 2^n 个点，在这之间的第 m 个点都是直角的顶点，m=1 的就是第一个直角顶点。

给出由 U 和 L 组成的折叠指令字符串（1≤长度≤30）以及整数 m，输出第 m 个点的坐标。



到处都是旅鼠（Lemmings, Lemmings Everywhere. But Not For Long, North America - East Central NA 2007, LA3966）

在一个 n 行×m 列（$n,m \leqslant 100$）的棋盘上，每个格子上都有只旅鼠。每只旅鼠都有一个由 4 个字母（NESW，表示北东南西）之一排列出来的移动计划。每一秒钟，旅鼠都尝试往东西南北其中之一的方向移动，规则如下：

（1）一开始每个旅鼠 L 都把方向 D 设置成其方向计划上的第一个方向。

（2）每一步 L 都尝试沿着方向 D 移动，可能发生 3 种情况：

- 如果 L 走出棋盘，那么旅鼠消失。
- 如果 L 的目标格子是空或者即将为空（上面的旅鼠离开），并且没有其他旅鼠要移到上面，L 就移过去并且方向依然为 D。
- 如果其他旅鼠要移到 L 的目标格子，或者上面有一个无法移动的旅鼠。那么 L 就原地不动，并且更新方向为其计划上的下一个方向（计划是环状的，需要时回到第 0 个字符）。

两个旅鼠是可以交换位置的，当然如果有其他旅鼠要移到它们所在的格子就不行（此时它们 3 个都需要停下不动）。在全部走出棋盘之前，他们会不停地移动。

棋盘上(0,0)坐标表示西南角，$(0,m-1)$表示东南角。按照从西到东、从北到南的顺序给出每个旅鼠的方向计划。计算并输出经过多少秒它们会全部消失。

淹没（The Flood, North America - East Central NA 2009, LA4537）

一个岛屿可用 $n \times m$（$n,m \leqslant 100$）的网格来表示，给出每个方格的海拔（≤1000）。方格只有垂直或水平连接才算相邻。海拔为 0 的格子（表示海水）都连接到岛屿的边缘。其他格子都联通。计算海水要上涨多高才能把岛屿淹到变成两块以上。或者有可能涨多高都不行，就输出“Island never splits.”。

猜数字游戏（Cover Up, North America - East Central NA 2009, LA4534）

有个猜字游戏，要猜一个 n（$n \leqslant 5$）位的数字，每位都给出一些备选数字。每一轮参与者针对每一位未猜中的数字在对应的备选数字中选择一个未选过的，如果至少猜中一位，则进入下一轮，猜所有未猜中的数字。如果全部都已猜对（赢）或者一位都没猜对，则游戏结束（输）。

对每一位，给出备选数字的个数 m，其中的 l（$0 \leqslant l < m \leqslant 10$）个数字给出正确答案在其中的概率 p（$0.0 \leqslant p \leqslant 1.0$）。例如，第 n 位有 5 种可能：1、3、5、8 和 9。其中正确答案是 5 或 9 的概率是 70%，那么 5 和 9 分别正确的概率是 35%，其他数字就是 10%。一开始猜 5 并且错了，但是其他位置有猜对的。那么接下来 9 的正确概率就是约 54%，大约其他数字约 15%。

计算出使用以上策略赢得游戏的概率，保留小数点后 3 位，不包含结尾的 0。必胜直接输出 1。

GIF 的解压算法（Decompressing in a GIF, North America - East Central NA 2009, LA4535）

下面是 GIF 图形编码压缩算法的简化版本。压缩过程中要维护一个字典，包含了不同字符串的十进制数字编码，不同字符串的编码长度必须相同。例如，26 个字母的编码就是 (A,

00),(B, 01),(C, 02),⋯,(Z, 25)。

压缩算法是一个循环，每次循环包含两个步骤：

- 遍历字符串未压缩部分的所有前缀，找到包含在字典中最长的那个，记为 p，把它替换成 p 对应的数字编码。
- 如果字符串还有未被压缩的部分，往字典中添加一个(s,n)映射，其中 $s = p +$（p 之后的未压缩部分的第一个字符），n 是字典还未用到的最大数字。

举例来说，字符串为 ABABBAABB，初始字典包含(A,0)和(B,1)两项。压缩过程每一步如表 4.2 所示。

表 4.2

字　符　串	最 长 前 缀	替 换 编 码	新增字典项
ABABBAABB	A	0	(AB,2)
0BABBAABB	B	1	(BA,3)
01ABBAABB	AB	2	(ABB,4)
012BAABB	BA	3	(BAA,5)
0123ABB	ABB	4	–

最终的压缩结果就是 01234。

输入压缩后的字符串，初始的字典大小 n（$1 \leqslant n \leqslant 100$）以及字典的内容。解压这个字符串并输出结果。

窗口（Windows, North America - East Central NA 2009, LA4540）

Emma 的电脑屏幕分辨率是 $10^6 \times 10^6$。打开了 n 个窗口，每个窗口都用 4 个整数 r、c、w、h 描述，其中（$r, c, 0 \leqslant r,c \leqslant 999999$）是窗口左上角像素的位置，（$w, h, 1 \leqslant w,h$）是窗口的宽高像素数。窗口输入的顺序就是 Emma 打开的顺序，后打开的窗口会覆盖之前的。输入保证窗口都完全包含在屏幕区域内。给出 m 个查询，每个查询包含一个点的坐标(cr, cc)，计算如果单击这个点会激活的窗口的序号 k，并且输出“window k”，如果不能激活任何窗口，则输出“background”。

需要注意的是，查询一个点不会真的单击激活对应的窗口。

翻转卡片（Flip It!, North America - East Central NA 2010, LA4901）

有一个 n 行 m 列（$n,m \leqslant 20$）的网格，每个格子上有一张牌。可以进行 4 种翻转操作：

- T 把最上一行的牌翻转了之后叠到第二行，如果格子上有一堆牌，就把它作为一个整体反过来叠到下面一行。
- B 把最下一行的牌翻转之后叠到倒数第二行，翻转规则同上。
- L 把左数第一列的牌翻转之后叠到左数第二列，翻转规则同上。
- R 把右数第一列的牌翻转之后叠到右数第二列，翻转规则同上。

对于每个格子，输入 $k>0$ 表示数字为 k 的面朝上的牌，$-k$ 表示面朝下。输入长度 $n+m-2$ 的一个翻转操作序列，包含以上 4 种操作。按照从底到上的顺序，输出所有面朝上的牌的编号。



拍照角度（Photo Shoot, North America - East Central NA 2010, LA4903）

Adam 有一个视角为 f（$0<f\leqslant180$）的相机，意思是如果朝从 x 轴左旋 d° 的方向拍照，$[d-f/2, d+f/2]$范围都会被拍下来。Adam 的位置是(x,y)，周围有 n（$n\geqslant0$）个人，其中$|x|,|y|,n\leqslant100$。对于每一个人 i，给出其坐标(x_i,y_i)，$|x_i|,|y_i|\leqslant1000$。 输入保证，包括 Adam 在内的所有人位置唯一。同时相对于 Adam 的位置，没有任何两个人的夹角刚好是 f。

计算输出如果需要保证所有的人至少被拍到一次，Adam 最少需进行几次拍照。

选举（Pro-Test Voting, North America - East Central NA 2010, LA4905）

Bob 要竞选市长。选民所在城市分为编号为 0～n−1 的 n 个选区，每个区都给出其人口 N（N<10000）。现在 Bob 希望通过投入费用来增加每个区投他票的选民比例，他可以投入的总费用是 m（$m\leqslant100$），对于编号为 p 的选区来说投钱的效果如下：

$$F_p = I_p + \left(\frac{M}{10.1+M}\right)\Delta$$

其中，I_p 是人口中的选民比例，M 是在这个选区投入的费用（以美元为单位），F_p 是能够达到的选民比例，Δ是百分比能够增加的最大值。

计算 Bob 如何花钱才能让所有选区中投他票的选民数量总和最大，并且输出每个区域的费用。在计算每个区域的选民增加数量时，需要先根据上文的公式计算出 F_p 的值，再根据人口基数乘以 F_p 后四舍五入输出。

中介（The Agency, North America - East Central NA 2011, LA5780）

未来的星际旅行中，每个行星用一个长度为 N（$1\leqslant N\leqslant1000$）的二进制串来表示。如果两个行星只有 1 位不同，则之间有航线直达。此航线的费用等于在目标行星降落的费用。如果这个行星的第 i 位是 1，必须交第 i 个税。所以费用总和就是对应税费之和。

给出出发行星 S 和目的行星 T，以及每一位对应的税费（1≤税费≤1000000）。输出从 S 到 T 的最小费用。

傻瓜链（Chain of Fools, North America - East Central NA 2011, LA5781）

图灵的自行车链轮上有编号为 0～s−1 顺时针排列的 s（$1<s<100$）个齿，其中最顶端的是 0，链轮是顺时针旋转的，下一个转到顶端的是 s−1，如图 4.21 所示。编号为 p 的齿已经损坏。

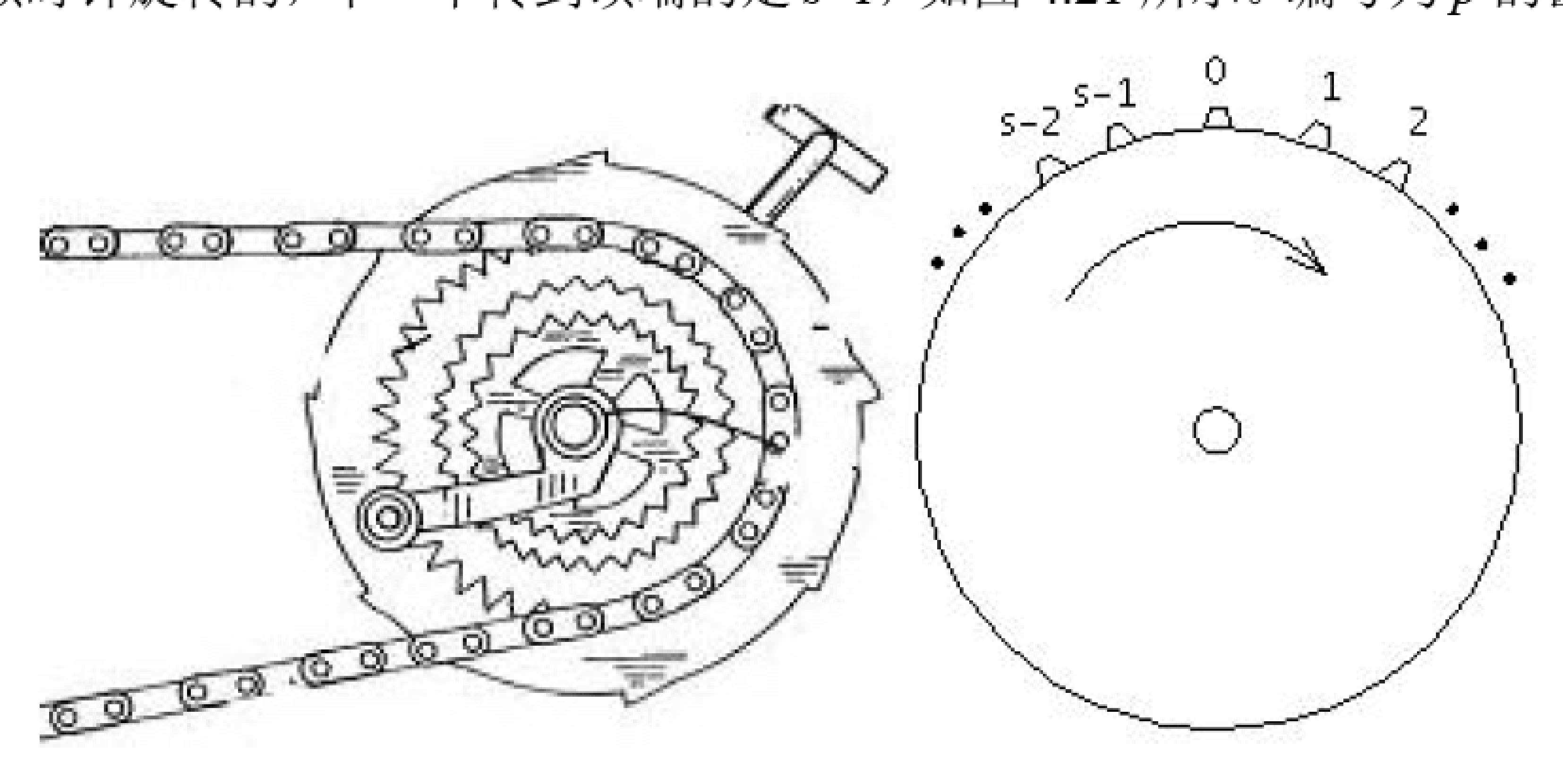


图 4.21

类似的，链子上也有顺时针排列的编号为 0～c−1 的 c（200>c>s）个链条，0 在最顶端，顺时针旋转，其中编号为1的已经弯了。如果坏齿和弯链同时转到位置 0，则链条脱落。输入的 p 和 1 不会是同时为 0。计算在多少转之后链子会脱落。如果会脱落，则输出 r m/s。表示需要 $r\frac{m}{s}$ 转。分数无须是最简分数，r 和 m 一定要输出，即使为 0。如果永远不会脱落，输出“Never”。

董事会决议（The Banzhaf Buzz-Off, North America - East Central NA 2011, LA5784）

在董事会中，每次要通过一个决议，都需要超过指定的票数。如果一组人的票数加在一起可以保证通过，那么称这组人为一个必胜联盟。其中如果有人离开就无法保证必胜，这个人就是关键的。举例来说，如果需要 26 票，必胜联盟可能就是 20、10 和 1。其中前面两个都是关键的（他们的离开会导致联盟只剩下 11 或 20 票）。

在权重分别为 20、11、10、8、1 的联盟中，如果决议需要 26 票通过，任何人都不关键；但如果决议需要 50 票，任何人都关键。对于任何一个成员来说，其 BPI 就是包含他并且以他为关键成员的联盟个数。

举例来说，如果一个决议需要 26 票，每个成员的 BPI 分别是 12、4、4、4、0。权重 20 的成员在 12 个不同的必胜联盟中是关键的。权重 11 的那个在 4 个不同的必胜联盟中是关键的。而且票数 1 的那位就完全没有任何权力。如果决议需要 42 票，那么 BPI 就变成 3、3、3、1、1。此时，权重 1 和 8 的二人权力是一样的。

输入不同的权重个数 n（1≤n≤60）以及决议通过所需票数 q，然后输入 n 对正整数：$w1$ $m1$ $w2$ $m2$…wn mn。其中，wi 是权重，mi 是权重为 wi 的成员个数。总的权重 V = $w1*m1+w2*m2+\cdots+wn*mn$ 满足 1≤V≤60，且 $V/2<q<V$，且 $i\neq j$ 时 $wi\neq wj$。计算输出每个成员的 BPI。

GPS 我爱你（GPS I Love You, North America - East Central NA 2011, LA5785）

路网中有编号为 0～n−1 的 n（n<100）个结点，给出任意两点之间的道路长度（如果为 0 则表示两点间没有道路，长度≤100）。然后给出 m 个结点，$p1,p2,p3\cdots pm$，按顺序表示一条风景比较好的从 $p1$ 到 pm 的路线。

如果让 GPS 导航仪来规划 $p1$ 到 pm 的路线，可能就是规划出一条最短路径，但是不一定风景较好。可以通过给导航仪按顺序输入一些强制的有向道路，使得 GPS 也导航出风景好的路线。可以假设如果两点之间有多个最短路径，导航仪会选择风景最好的那条。

计算如果要让导航仪导航出同样的路径，最少需输入多少个强制的有向道路。

快闪党（Flash Mob, North America - East Central NA 2012, LA6123）

在一个城市中，城区是一个完美的网格，所有的街道都是东西或南北向的，并且平行街道之间的间距都是 1。快闪党有 n（1≤n≤1000）个成员，每个成员都在某个街道的交点处，且只能沿着街道水平或者垂直移动。给出成员的坐标（0≤坐标值≤106）。计算出一个街道交点作为召集点，使得所有成员到达这个点移动的距离之和最小。如果有多个答案，计算出坐标值字典序最小的那个。输出这个点的坐标以及所有成员移动到这个点的距离之和。



例如，有 5 个成员，坐标分别是(3, 4)、(0, 5)、(1, 1)、(5, 5)和(5, 5)。如果把它们召集到(3,5)，最小的距离之和就是 14。

数路牌游戏（Road Series, North America - East Central NA 2012, LA6127）

Don 和 Jan 二人喜欢在路上玩一个游戏，目标是在看到的牌子上找到数字 1，接着是 2，接着是 3…。到两位数字时，在牌子上找到的两位必须是紧跟着的，并且任何一个牌子都可以提供多个数字。例如，如果看到一个牌子上有字符 678−43 15，那么可供使用的数字就是 67、78、43、15，但是 84 不行（被破折号分开），被空格分开的 31 也不行。当然单独的数字 6、7、8、4、3、1、5 以及 3 位数 678（如果能一直玩到这么大）都是可用的。

他们把数字 *n* 称为最大完成数，如果 *n* 是所有满足“1～*n* 都已经发现”这个条件的数字中最大的。一开始 0 是最大完成数。这样就可在 *n* 之外记录所有见到的数，只要他们不比 *n* 大太多。

在他们的游戏中，用一个滑动窗口来记录所有见过的数字，这个窗口的大小是 *w*，这样他们可以记住小于等于 *n*+*w* 的任何数字。

例如，*w* = 4，当前最大完全数是 19，当看到如下的标牌：

Show time at 8:25, no one under 21 admitted

那么他们就可以记录 21，但是不是 25（不在滑动窗口内），如果标牌如下：

The FleaBag Hotel, phone 555−2520

他们就可以用 20，然后 21 会变成最后完成数，接着就可以用 25，因为现在 25 就被窗口覆盖了。

给出 *k*（*k*≤1000）个标牌的文字（包含数字、字母、标点和空格），长度都不超过 1000，以及滑动窗口大小 *w*（*w*≤100）。输出使用这些文字可以找到的最大完全数，以及最大数字。

货币兑换（Show Me the Money, North America - East Central NA 2012, LA6128）

Frank 手上有许多货币，以 val_1 $name_1$ = val_2 $name_2$ 的形式给出货币之间的 n 种汇率，其中 $name_1$ 和 $name_2$ 是不同的货币名称，名称不超过 10 个字母。总共有不超过 8 种货币，任意两种货币的汇率只会给出一次，并且不会出现自相矛盾的汇率（如 1A = 2B，1B = 2C 以及 1C = 2A）。

Frank 手上每种货币的数量都不超过 100000。假如要取一种货币但是他手上刚好没有，那么就需要按照汇率兑换成另外一种货币来代替，兑换的原则是超过并且尽量接近所需要的值。同时兑换的结果不能超过 100000。

例如，现有 6 种货币（A、B、C、D、E、F），汇率如下：

23 A = 17 B

16 C = 29 E

5 B = 14 E

1 D = 7 F

假设需要 100 面值的 A，可以使用的兑换方案是 74 B（≈100.12 A），115 C（≈100.72 A）或者 207 E（≈100.02 A），那么最优的兑换方案就是 207E，注意 Frank 无法计算出 A 和 D、

A 和 F 之间的汇率。因为需要使用超过 100000 的 E，Frank 也无法用 E 来代替 64000A，必须用 73078C 来替代。

给出 *n* 个兑换请求，每个请求形如 val name，表示需要 val（val≤100000）面值的 name 货币而刚好没有，必须用其他货币兑换。计算最优的兑换方案。

战舰游戏（Battleships, North America - East Central NA 2013, LA6553）

有一种战舰游戏，给出一个 10×10 的方阵，上面秘密地放了 10 艘船。其中有 1 个占 4 格，2 个占 3 格，3 个占 2 格，剩下的就占 1 个格子，互相都不重叠并且不能相邻，对角相邻都不行。对于战舰位置唯一的线索就是，在方阵下方和右方打印出的对应行/列上被占用的格子数量之和。下面是一个战舰游戏的方阵以及答案。

设计战舰游戏时，必须保证对于一个指定的行列和必须有唯一的答案（如图 4.22 所示）。但是有些时候，必须要指定若干个格子才能保证只有 1 个答案，如图 4.23 所示。

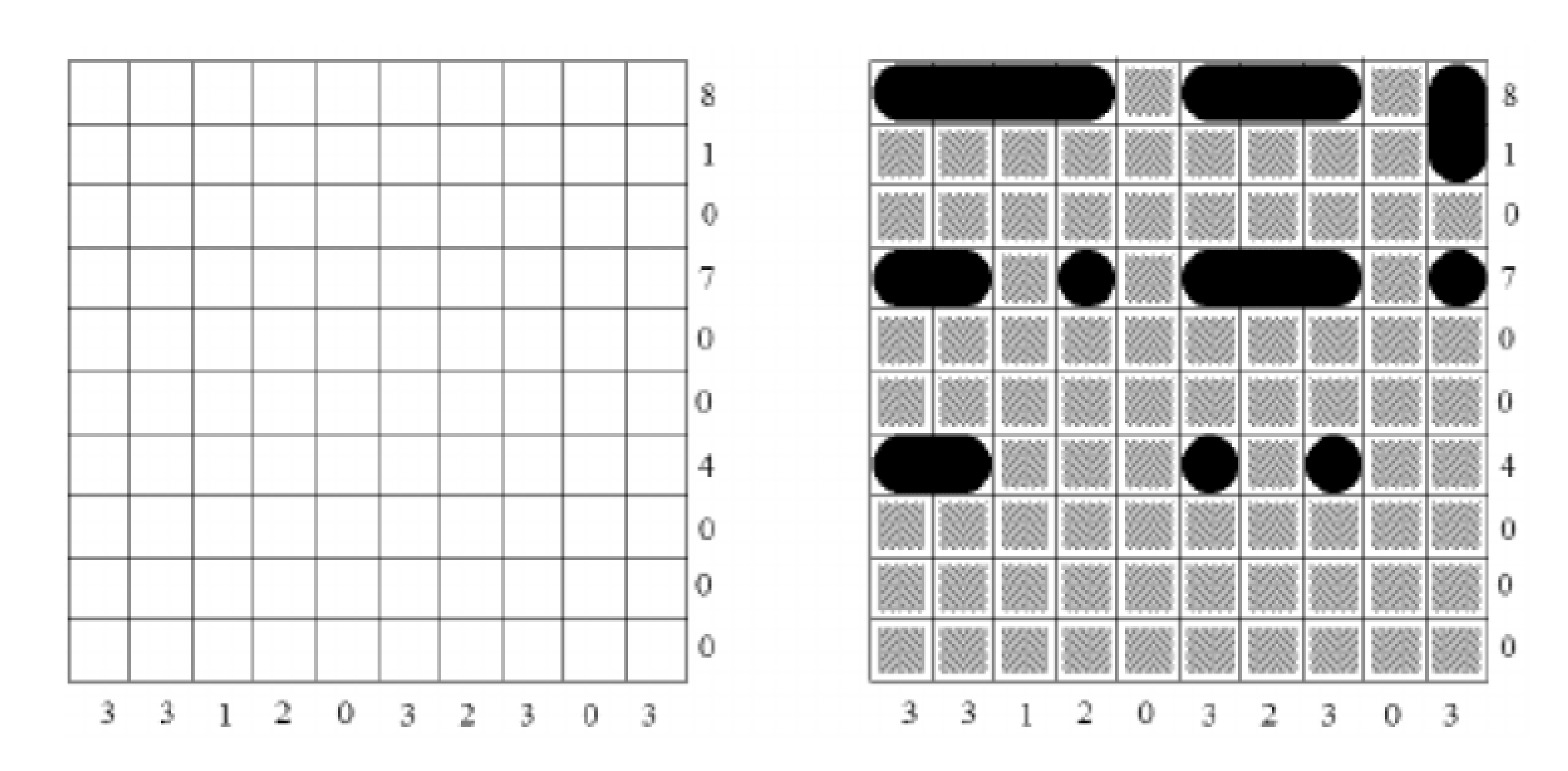


图 4.22

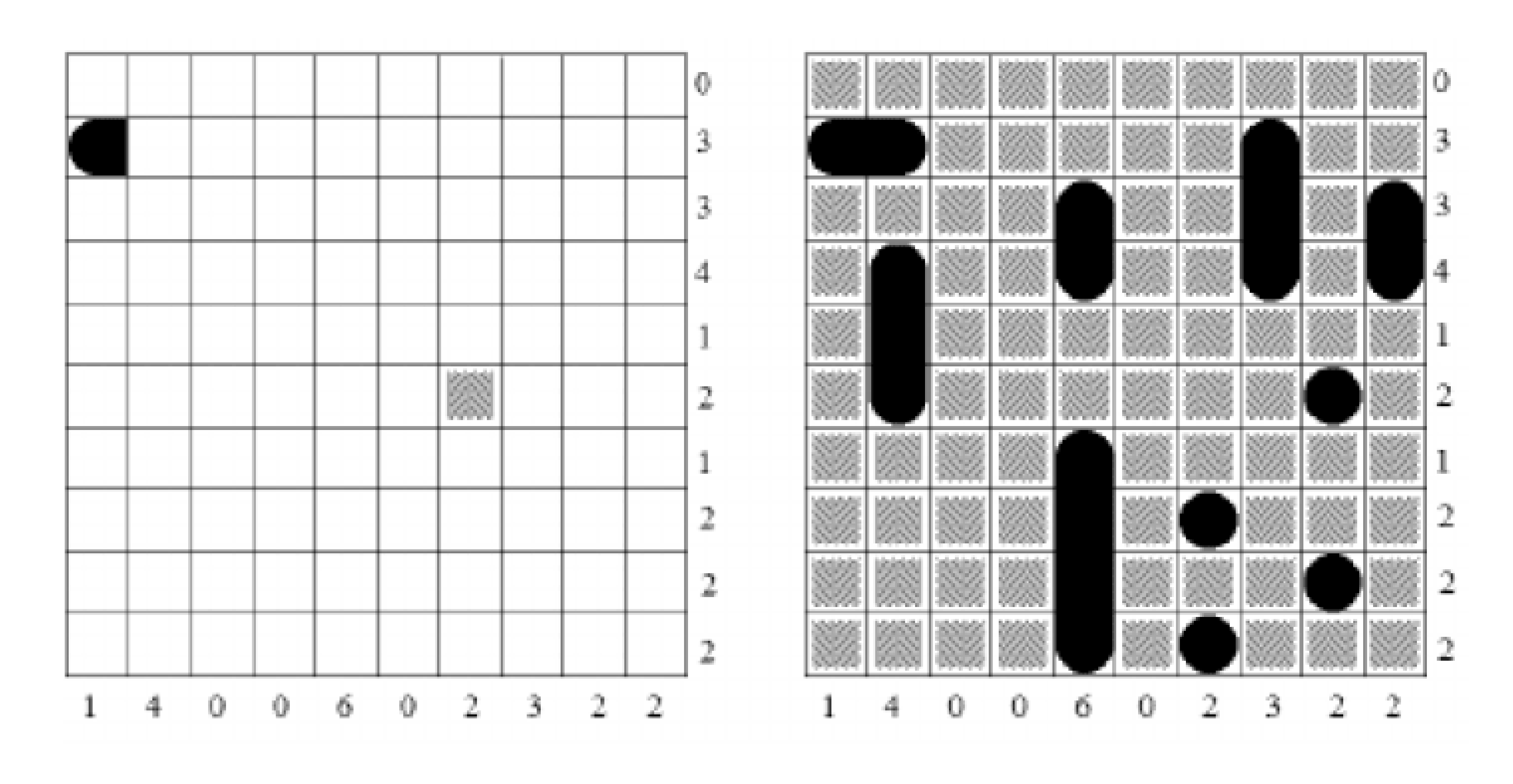


图 4.23

指定的格子可以是 7 种类型：水、船体中部、水平船的左端、垂直船的顶端、水平船的右端、垂直船的底端以及 1 格的船。分别用图 4.24 中的字符表示，其中 O 是大写字母，X 是大写，w、v 是小写。

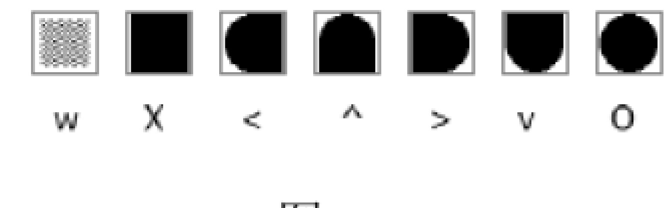


图 4.24

给出每个行列的和（都不超过 12000），计算保证游戏只有一个解所需要指定的最小的



方格个数。如果这个数字大于 2，那么直接输出“too ambiguous”。否则首先输出答案，然后输出需要指定的方格的位置以及字符。如果有多个答案，依次按照行最小，列最小，然后是按照上图中的顺序作为字典序输出答案。如果答案是两个方格，按照第一个格子，然后第二个格子最小的方案输出。行列都从 1 开始计数。

逃窜（Stampede!, Regional North America - East Central NA 2013 LA6557）

有一个 $n×n$（$n≤25$）的棋盘。除了左右两端的两列之外，某些方格可能包含障碍物。最左边的一列是你的 n 个棋子，每行 1 个。你的目标是要把你的棋子尽快地移到最右边那一列。每次可以把每个棋子往 N、S、E、W 4 个方向之一移动一步，或者原地不动。棋子不能移到包含障碍物的格子内，也不能一次把两个棋子移到同一格。所有棋子同时移动，可把一个棋子移到当前被另外一个即将移走的棋子占用的格子中。

输入 n 的值以及所有障碍物的位置，计算把 n 个棋子移到最右边一列所需要的最少的步数。输入保证左右两列之间一定有一条路上没有障碍物。

超级菲利斯（Super Phyllis, North America - East Central NA 2013, LA6558）

有一家跨国企业采用如图 4.25 所示的报告传递模式。

如果有人想给他的所有上级发报告，需要沿着箭头方向每条线都发一份报告。图中就是 D 发给 B，B 发给 A。其实 D 给 A 没必要发，因为报告内容已经由 B 发给 A 了。所以 D 到 A 的链接就可以删掉。如果 C 到 B 也有链接，那么 D 到 B 和 C 到 A 的链接也都可以删掉。

图 4.25

输入企业中每个员工的姓名，以及员工之间的汇报关系。输出可以被删除掉的链接数目，并且按照两端字母的字典序输出被删掉的所有链接。输入保证不超过 200 个员工并且没有重边。

ACM/ICPC North America - Mid-Central USA

魔法师的标记（The Mark of a Wizard, North America - Mid-Central USA 2012, LA6098）

地精在地下挖的隧道形成如图 4.26 所示的网络。垂直向上也是图中朝上的方向。巢穴在 A 点，地面出口用 F 表示。其他的标签表示隧道交点。这个图是一个 3D 隧道系统的平面示意图，所以看起来交叉的路径实际上并没有相交（如边 BD 和 CE）。

有一群魔法师希望能在探险时，能尽快回到地面。很多朝上的隧道都是从岔路口开始，并且岔路口无法选择到底哪边是最优路径。魔法师搞清楚了通过每条隧道所需的时间，就是图 4.26 中用数字做的标记。除此之外，他们还希望在每个岔路口做一些标记，告诉魔法师哪边到地面的时间最短，但是他们希望标记最少的岔路口，以防被发现。在图 4.26 中，一种可能的标记方案包含 A→B，接着是 B→C。从 C 只有一条朝上的路，无须标记。所以总的时间是 3+1+4=8，刚好是最短时间。

在图 4.27 中，就只有 1 个标记，E→D。这样不能决定唯一路径，但是保证了最短时间。从 A 出发，可以走 B 或 E。如果走 B，有两条朝上的路径都是最短时间就是 8。E 就必须标

记，否则就可能走出 AECF 这样总时间为 2+3+4=9 的路径。所以为了保证走出隧道一定用的是最少时间 8，至少需要 1 个标记。另外还有一种 1 个标记的方案，A→B。

给出结点的个数 n（$2\leqslant n\leqslant 17$），这些结点包括起点、终点以及隧道的交叉口。然后按照字母顺序输入 n 个用大写字母标签，以及从这些结点出发的路径个数 u。然后是 u 个路径的目标结点以及所需时间 time（1≤time≤500）。起点一定是 A，最后一个字母一定是出口。除了出口的 u=0，其他地点 $1\leqslant u\leqslant 6$。

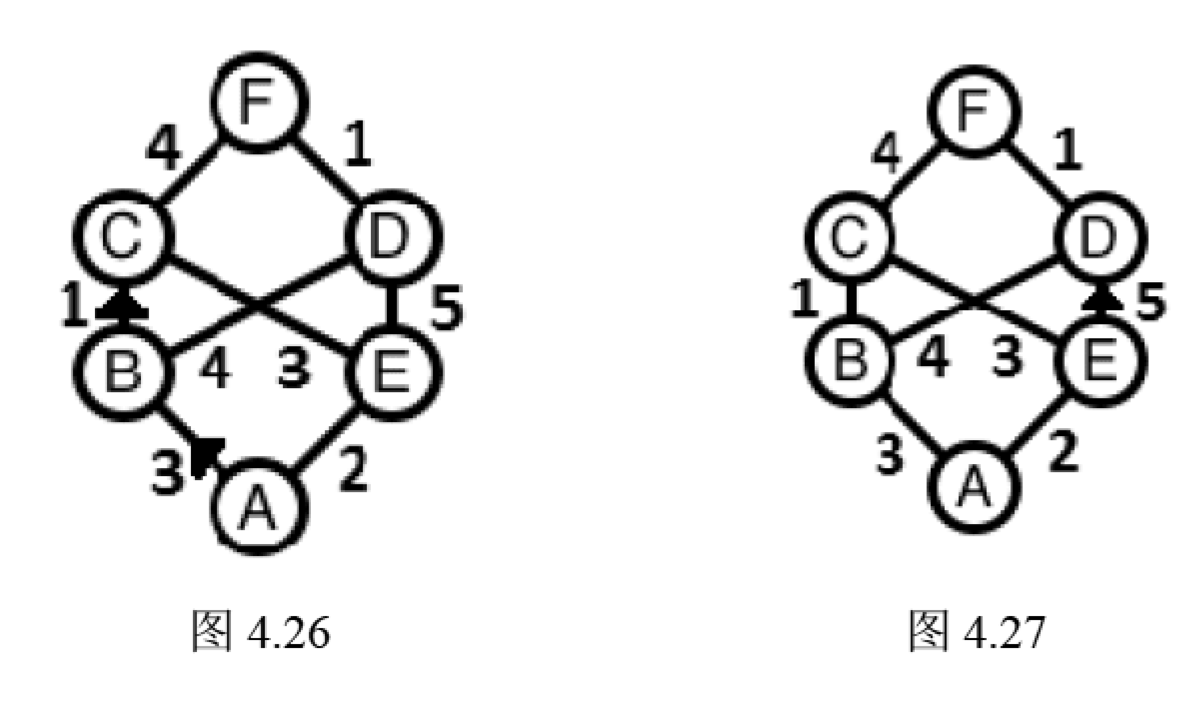


图 4.26　　　　图 4.27

在每一个结点，魔术师都只走朝上的路径。隧道的个数不超过 35。一定有一条从巢穴到地面的最短路径，这条路径最多包含 7 个隧道。除了起点和终点，每一个结点都包含出入隧道至少各 1 条。

输出从巢穴出发上升到地面所需要的最短时间，以及要保证所有魔法师都一定可以用最短时间走到地面所需要的最少的标记个数。

ACM/ICPC Latin America

名称分配（Dividing the names, Latin America 2014, LA6824）

S 城市是一个网格形状，有 N 条南北向的互相平行的大道，和 N 条东西向的互相平行的大街。每条大道都和每条大街相交。

现在已经有 $2\times N$ 个名称，需要分配给所有的街道，所有的交叉路口路牌都必须写上两条对应街道的名称。希望用缩写来表示街道名称，使得路牌上的字数尽量少。一个大街的缩写名称应该是名称字符串的不与其他大街缩写名称冲突的最短前缀。大道也是类似的原则。

例如，当 N=2，4 个名字是 GAUSS、GALOIS、ERDOS 和 EULER。如果把大街命名为 GAUSS 和 GALOIS，大道命名为 ERDOS 和 EULER。对应的缩写就是 GAU、GAL、ER、EU。根据这个分配方案，4 个交叉路口的路牌就是 GAU|ER、GAU|EU、GAL|ER、GAL|EU，总共需要 20 个字符。

然而有另一种方案，大街是 GAUSS 和 ERDOS，大道是 GALOIS 和 EULER。那么路牌就是 G|G、G|E、E|G、E|E，只有 9 个字符。

输入 $2\times N$（$2\leqslant N\leqslant 100$）个名称（长度不大于 18，全部是大写字母），保证不会有一个

名称是其他名称的前缀。计算路牌上字符个数的最小值。

平均分配（Even distribution, Latin America 2014, LA6825）

E 要带一些孩子去一个群岛旅行。群岛中有 I 个编号为 1～I 的岛，S 条连接两个岛的双向航线。E 的旅行路线可以从其中任意一个岛开始，在任意一个岛结束，只要一路都有航线，也可以路过一个岛多次。

E 在带着孩子到达岛 i 时，会收到 C_i（$1 \leqslant C_i \leqslant 10^5$）个糖果。虽然自己不吃，但是必须把这 C_i 个糖果给随身的孩子均分。这样 E 的旅行计划就决定了他能带的孩子的个数 k。

计算所有满足下述条件的 k 的个数：k 必须能够让 E 制订出一个旅行计划，使得 E 在收到每个岛上的糖果之后能均分成 k 份。

帮助丘比特（Help cupid, Latin America 2014, LA6828）

给出 N（$2 \leqslant N \leqslant 1000$ 且 N 是偶数）个单身男女，其中男女各 $N/2$ 个。对于一对男女来说，如果他们所在时区分别是 i 和 j（$-11 \leqslant i, j \leqslant 12$），那么他们的时区差就是 $\min(|i-j|, 24-|i-j|)$。给出 N 个人的时区，计算出一种男女配对方案，使得配对之后所有情侣的时区差之和最小。输出这个最小和。

勇敢的登山者（Intrepid climber, Latin America 2014, LA6829）

山上有 N 个地标，只有一个在山顶，你也在山顶。你有 F（$1 \leqslant F < N \leqslant 10^5$）个朋友都在其他某个地标处。并且你希望去访问他们，地标之间都有阶梯相连，从山顶通向每一个地标都有且只有一条路，要访问所有的朋友，你必须先下一些阶梯，再上，再下，再上。下坡不费体力，但是每次上一个阶梯都要花费一定的体力。访问完所有朋友之后就可以立刻坐下休息了。

图 4.28 中，N=6。你朋友分别在 2 和 5。你可以 1↓2↑1↓3↓5 的顺序来访问他们。其中，a↓b 表示从 a 下到 b，a↑b 表示从 a 爬到 b。另外一条可能的顺序就是 1↓3↓5↑3↑1↓2。

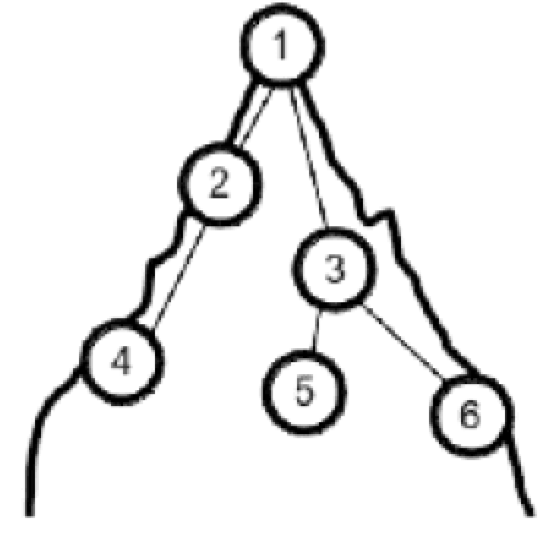


图 4.28

给出所有的阶梯，每个阶梯给出其连接的地标 A 和 B（$1 \leqslant A \leqslant B \leqslant N$，$A \neq B$），以及爬阶梯需要耗费的体力 C（$1 \leqslant C \leqslant 100$），给出 F 个朋友所在的路标位置。计算出从山顶出发访问完这些朋友所需耗费的体力之和的最小值。

圆桌武士（Knights of the Round Table, Latin America 2014, LA6831）

亚瑟王经常和他的武士围着圆桌举行会议，其中亚瑟王坐在王位上。围绕圆桌有 K 个按照顺时针标记为 1～K（$1 \leqslant K \leqslant 10^6$）的座位，王位的左边是 1 号。每个武士都有一个唯一的 1～K 之间的编号，坐在对应的位子上。一开始亚瑟王坐在王位上，D（$1 \leqslant D \leqslant 10^5$）个武士进来之后坐了 D 个座位，但是其中某些人坐错了。之后 $K-D$ 个武士就要依次按照以下的规则入座：

（1）尽量坐在自己编号对应的位置。

（2）如果这个位置被占了，那就按照顺时针寻找下一个没被占的位子。

最后所有武士的座位布局就跟他们到达的顺序有关。现在给出前 D 个人的位置。亚瑟

王需要你计算出最终 K 个座位可能布局的数目，输出其模 10^9+7 的余数。

车攻击（Attacking rooks, Latin America 2013, LA6525）

一个大小为 $N\times N$（$1\leqslant N\leqslant 100$）的棋盘，有的格子放了卒，这些格子不能再放车。两车必须在同一行或者同一列，且中间没有卒，才能互相攻击。给出卒的位置，计算在不相互攻击的前提下，最多能往棋盘上放多少车。

足球（Football, Latin America 2013, LA6530）

足球比赛中，赢一场球得 3 分，输得 0 分，平局双方各得 1 分。N（$1\leqslant N\leqslant 10^5$）场比赛之后，球队有输有赢，可以花钱买一些进球数，加到过去的比赛中来改变已有的结果。

给出每一场比赛的进球数 S 和被进球数 R（$0\leqslant S,R\leqslant 100$）。计算出在最多可以买 G（$0\leqslant G\leqslant 10^6$）个进球的前提下，总的得分能够被修改到的最大值。

爬上顶点（Go up the Ultras, Latin America 2013, LA6531）

对于海拔为 h 的山顶 p 来说，其突出度定义为：

（1）如果有海拔更高的山顶，那么从 p 到其他任何一个山顶都要经过一个海拔 $h-d$ 的最低点。那么突出度就是所有这些 d 中的最大值。

（2）否则就是 h。

在图 4.29 中极点就是 7、12、14、20 和 23。

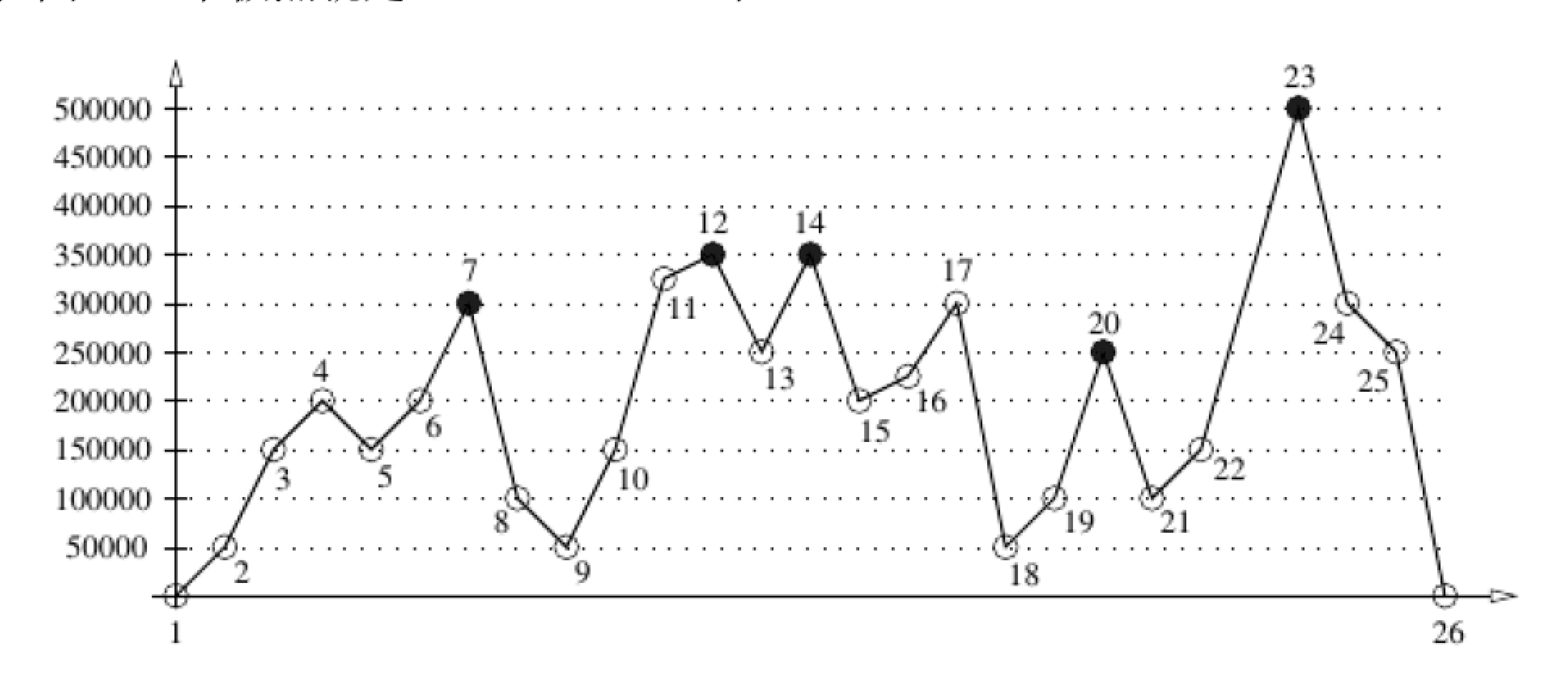


图 4.29

如果突出度高于 150000cm，就称这个山顶为极点。给出编号为 1～N 的 N（$3\leqslant N\leqslant 10^5$）个山顶，假设都在一条直线上，且相邻山顶的海拔都不同。山顶的高度为 H_i（$0\leqslant H_i\leqslant 10^6$，$i=1$～$N$）。其中，$H_1=H_N=0$。计算并输出其中所有极点的编号。

反相哈夫曼（Inverting Huffman, Latin America 2013, LA6533）

哈夫曼编码算法生成树的过程中，权值最小的树可能不止两个，若选择其中不同的两棵树最终会产生不同的二叉编码树。

假如待压缩文本中不同字符个数是 N（$2\leqslant N\leqslant 50$）。对于每个字符 i，给出算法生成的编码长度 L_i（$1\leqslant L_i\leqslant 50$，$i=1, 2, \cdots, N$）。计算要生成这样编码长度的编码树，待压缩文本的最短长度。

连接两国（Join two kingdoms, Latin America 2013, LA6534）

N 国和 Q 国各有编号 1～N 的 N 个和编号 1～Q 的 Q 个城市（$1 \leqslant N, Q \leqslant 4\times10^4$）。每个国家都有双向路网，其中每条路连接两个城市，且任意两城间路径唯一。定义任意两城之间路径长度之最大值为路网的大小。

要在两国中各选一城，等概率随机选择，连接起来之后形成一个新的路网。计算其大小的期望值。

ACM/ICPC SWERC（Southwestern Europe Regionals）

不给糖就捣蛋（Trick or Treat, Regional SWERC2009 LA4504）

给出平面上 n（$1 \leqslant n \leqslant 50000$）个位置唯一的点的坐标，其坐标值$(x,y)$都满足（$-200000 \leqslant x, y \leqslant 200000$），以米为单位。在 x 轴（$y = 0$）上找一个点，使得这个点到 n 个点的距离的最大值最小。输出这个点的坐标，以及这个点到 n 个点的距离的最大值。输出保留小数点后 9 位，可以允许不超过 10^{-5} 的误差。

装配线（Assembly line, Regional SWERC2010,LA4961）

有一系列的零件需要组装，但是不同的组装顺序所需要的时间可能不一样。每次只能组装相邻的两个零件，成为一个新零件继续和其他零件组装。

输入所有零件两两组装所需要的时间以及组装的结果。例如，有两种零件$\{a,b\}$，则输入表如图 4.30 所示。

	a	b
a	3-b	5-b
b	6-a	2-b

图 4.30

意思是：a,b 组装需要 5 分钟，生成一个 b。接着 b,a 组装需要 6 分钟再生成一个 a。注意这个表是不对称的，组装 b,a 和 a,b 需要的时间和生成结果都不同。

对于一个零件序列 aba，两种组装顺序需要的时间分别是：

（1）$(ab)a = ba = 5+6 = 11$。

（2）$a(ba) = aa = 6+3 = 9$。

最优的组装时间是 9 分钟。

输入 k 个小写字母表示的零件，然后两两输入每一对零件组装所需要的时间 t（$0 \leqslant t \leqslant 1000000$）和组装结果。输入一个长度为 n（$n \leqslant 200$）的待组装的零件序列，计算将这些零件全部组装完成所需要的最短时间。

三面骰子（3-sided dice, Regional SWERC2010, LA4963）

需要模拟一个三面的骰子，使其扔出 3 个面（编号 1,2,3）的概率都等于指定的值。模拟过程如下：给出 3 个三面骰子，每次随机选择一个，扔出去之后报告它的值。你可以指

定选择特定骰子的概率，但要求这个概率大于 0。

给出 4 个骰子，每个骰子用 3 个在[0,10000]区间内和为 10000 的整数描述。表示扔 10000 次结果分别为 1、2、3 的次数。其中第 4 个骰子表示所要模拟的结果。输出是否能用前面 3 个筛子模拟出第 4 个。

在测试案例 1 中，需要模拟一个结果 1,2,3 的概率分别是 $\frac{3}{10}$, $\frac{4}{10}$, $\frac{3}{10}$ 的骰子。在这个案例中给出的骰子中扔第 i 个的结果永远是 i（i=1,2,3）。那么就可以这么模拟：用 $\frac{3}{10}$ 的概率扔 1，$\frac{3}{10}$ 扔 2，$\frac{4}{10}$ 扔 3。

样例输入：

```
0 0 10000
0 10000 0
10000 0 0
3000 4000 3000

0 0 10000
0 10000 0
3000 4000 3000
10000 0 0

0 0 0
```

投票箱的分发（Distributing Ballot Boxes, Regional SWERC2011, LA5822）

大选开始了，要分发投票箱，规则如下：

（1）有 N（1≤N≤500000）个城市，每个城市发至少 1 个箱子，共有 B（N≤B≤2000000）个投票箱。

（2）第 i 个城市人口为 a_i（1≤a_i≤5000000）。

（3）每个人都要在他/她分配的箱子中投票。

（4）要让所有箱子中，单个箱子分配的人口数量的最大值最小。

依次输入 N 和 B 以及每个 a_i，计算出符合以上规则的最优分配方案，并输出其中单个箱子分配的人口数量的最大值。

第 1 个案例中：第 1 个城市放两个箱子，剩下的箱子放到第 2 个城市，然后正好每个箱子中分配了 100000 个人口。

第 2 个案例中：城市中分配的箱子分别是 1 2 2 1，第 3 个城市中每个箱子中会有 1700 个人去投票，是最优方案中单个箱子分配人口数量的最大值。

样例输入：

```
2 7
200000
500000
```

```
4 6
120
2680
3400
200
-1 -1
```

样例输出：

```
100000
1700
```

结对检查（Peer Review Regional SWERC2011, LA5826）

在一次科学会议上，科学家要互相检查论文，规则如下：

（1）每篇论文都要由多人检查。

（2）不能检查自己的或者跟自己合作的作者的论文。

（3）一个人不能多次检查同一篇论文。

输入每篇论文被检查的次数 K（$2 \leqslant K \leqslant 5$）以及论文个数 N（$1 \leqslant N \leqslant 1000$）。论文都只有一个作者，并且每个作者只能提交一篇论文。

接下来的 N 行，每一行包含了一个作者及其所属的机构。接着是这个作者被要求检查的 K 篇论文。可以假设来自同一机构的作者都有合作，不同机构的作者没有合作。机构名称是由字母组成的长度不超过 10 的字符串。论文是以他们作者的编号来命名。论文 1 的作者就是输入列表中的第 1 个，论文 N 是最后一个。

计算检查规则总共被违反了多少次。

颜色混合（Mixing Colours, Regional SWERC2013, LA6570）

有一种游戏，初始给出一些不同颜色的牌子。每一对相邻的牌子如果颜色符合特定的规则，就可以组合（顺序无关）在一起变成另外一个颜色的一张牌子。这样持续下去，直到只剩一张牌子为止。但是不是每一对不同的颜色都可以转换。例如，给出如下的规则：

蓝+黄→绿

黄+红→橙

蓝+橙→棕

就可以这样转换，（蓝，黄，红）→（蓝，橙）→（棕），游戏结束。

但是游戏中有些牌子的颜色看不清了，只有一定的概率确定是某种颜色。给出 R（$0<R \leqslant 100$）个转换规则，每个规则都给出相关的 3 个颜色的名称。然后给出长度为 C（$0<C \leqslant 500$）的颜色序列表示 C 个牌子。对于序列中的每一项，给出一系列的概率描述(k, cer(k))，表示这个牌子颜色为 k 的概率为 cer(k)（$0<\text{cer}(k) \leqslant 1.0$）。序列以 END 结束。对于一个特定的令牌，所有颜色的概率之和为 1.0。对于组合 $A+B \to C$ 来说，获得颜色 C 的概率是 $\text{cer}(C) = \text{cer}(A) \times \text{cer}(B)$。

输出游戏最有可能的结果以及这种结果的概率。

输入样例解释：

样例 1：只有两个牌子，而且第 2 个确定是黄色的，但是第 1 个可能是红或橙。所以游戏的可能的两种结果是：

（红, 黄）→（橙）：概率 0.7。

（橙, 黄）→（黄）：概率 0.3。

最终答案是橙：0.7。

样例 2：有多个牌子和不同的概率：

（1）（蓝,黄,黄,红）→（蓝,黄,橙）→（蓝,黄）→（绿）：概率 0.006。

（2）（黄,红,白,黑）→（绿,粉,黑）→（绿,红）→（黄）：概率 0.036。

最终答案是黄：0.036。

样例 3：只有两个颜色确定的牌子。而且没有任何转换规则，所以无解。

样例输入：

```
7
Blue Yellow Green
Yellow Red Orange
Green Red Yellow
White Red Pink
Pink Black Red
Orange Red Red
Yellow Orange Yellow
3
2
Red 0.7 Orange 0.3 END
Yellow 1.0 END
4
Blue 0.6 Green 0.4 END
Red 0.2 Orange 0.6 Yellow 0.2 END
White 0.9 Yellow 0.1 END
Red 0.5 Black 0.5 END
2
Blue 1.0 END
Orange 1.0 END
```

课程安排（It Can Be Arranged, Regional SWERC2013, LA6571）

学校每天有 N（$1 \leqslant N \leqslant 100$）种课要上，每种都各有一节。对于第 i 个课程来说，时间范围是$[A_i,B_i]$（$0 \leqslant A_i \leqslant B_i \leqslant 10000000$），有 S_i（$1 \leqslant S_i \leqslant 10000$） 个人来上这个课，一个人每天只上一节课。每个教室的容量是 M（$1 \leqslant M \leqslant 10000$）。如果一节课的人数大于 M，那么就要用多间教室。

给定了一个矩阵 clean(ij)，表示的是上完 i 课程需要 $clean_{ij}$ 的时间打扫卫生才能继续上

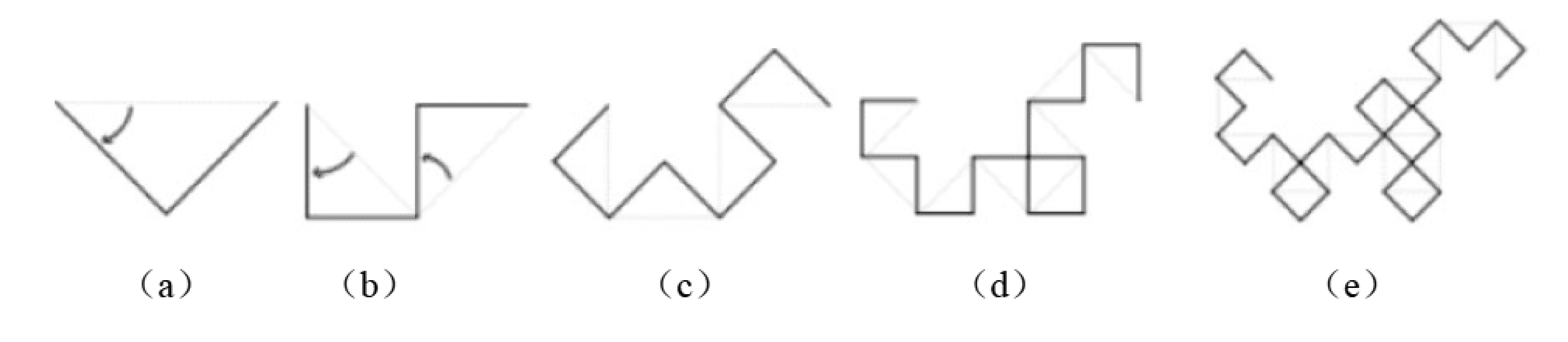


j 课程。也就是说，一间教室如果上完 i 课程要上 j 课程就需要满足条件 $B_i+\text{clean}_{ij}<A_j$（$1\leqslant i\leqslant N$，$1\leqslant j\leqslant N$，$0\leqslant \text{clean}_{ij}\leqslant 10000000$，$\text{clean}_{ii}=0$）。

计算最少需要多少间教室才能满足所有课程需要。

走廊解码（Decoding the Hallway, Regional SWERC2013, LA6573）

有一条路，有个法师从左边进去，从右边出来，这样做 n 次。每次走的过程中都要把路的每一段交替地向左右弯曲，如图 4.31 所示。



（a）　（b）　（c）　（d）　（e）

图 4.31

（1）第 1 次（如图 4.31（a）所示）：开始是条直线（图中的虚线）。所以就把这一段弯到右边（从左到右走，形成图中的实线）。

（2）第 2 次（如图 4.31（b）所示）：这次路已经变成 2 段。所以就把第一段弯到右边，第二段到左边（图中的实线）。

（3）第 3 次（如图 4.31（c）所示）：路是 4 段，接着就把四段向右、左、右和左弯。

（4）如此往复下去，图中就是做了 4 次和 5 次的结果。

如果有一个人从左边进去从右边出来，走的过程中记下每一次拐弯，右拐记 R，左拐记 L，形成一个轨迹字符串。所以如果 n=1，结果是 L。n=2，结果是 LLR。n=4：LLRLLRRLLLRRLRR。给出一个由'L'和'R'组成的字符串 S（$S\leqslant 100$）以及 n（$n\leqslant 1000$）的值。判断 S 是否是走 n 次之后的轨迹字符串的连续子串。

二叉树（Binary Tree, SWERC2013, LA6577）

给出一颗无限大的完全二叉树，以及一个由 L、R 和 U 组成的指令字符串，L 表示向左，R 表示向右，U 表示向上。把当前位置设置为根结点，接着按照输入的指令串 S，L 就移到左叶子，R 是右叶子，U 就到父结点（根结点的父结点就是自己）。

接着是另外一个指令串 T。对于 T 来说，可以忽略其中的任意指令（也可以全部忽略），我们需要计算在执行 T 中的指令（任意忽略）之后，所在位置可能有多少个不同的值。

例如，S = L，T = LU。答案是 3。执行 S 之后，我们在根结点的左叶子。接着执行 T，有 4 种可能：

（1）忽略所有指令，位置不变。

（2）忽略 L，执行 U，在根结点。

（3）执行 L，忽略 U，会在当前结点的左叶子。

（4）执行 L 和 U，位置不变。

所以总共可能有 3 个位置。

输入 S 和 T（长度都不超过 100000）。计算按照上文所述规则执行 S 和 T 之后，所有可能所在位置的个数，输出其模 21092013 的余数。

ACM/ICPC Europe - Central

无聊的扑克牌游戏（Boring Card Game, Europe - Central 2011, LA5879）

有种多轮的扑克牌游戏，N（$1 \leqslant N \leqslant 1000$）个编号为 $1 \cdots N$ 的玩家从左到右坐成一排。牌桌上有编号 $1,2 \cdots 5N$ 的 $5 \times N$ 张牌。

游戏一开始，进行 3 圈发牌：

（1）从左到右每人从牌堆顶端取出 2 张牌。

（2）也是如此。

（3）从左到右每人发 1 张牌。

之后每个玩家拿到 5 张牌，如果谁拿到的牌刚好编号是最小的 5 个（1,2,3,4,5 顺序无关），那么他就赢了这轮。

如果没有赢家，从右到左收回每个玩家的牌。对于每个玩家来说，收牌的顺序和一开始发牌的顺序相反。每张牌都放到牌堆顶上，然后下张牌也放到顶上。收完之后，顶上的牌就是玩家 1 的牌，并且 6 张牌在原来牌堆中的位置依次是 $1,2,2N+1,2N+2,4N+1,3$，然后再重新开始游戏。

给出一开始牌堆每张牌的顺序，计算一轮游戏的最终结果。如果玩家 P 在第 G 轮赢，输出“Player P wins game number G.”，游戏从第 1 轮开始编号，G 有可能超过 2^{32} 但是不会超过 2^{63}。如果游戏永远无人能赢，输出“Neverending game”。

地铁（Subway，Regional Europe - Central 2013 LA6583）

Jonny 要乘地铁到朋友那里，他特别喜欢坐地铁，所以希望坐尽量长的距离。但是他爸爸希望他换乘的次数最少。地铁任意的相邻两站之间的行驶时间都是 1 分钟，换乘可以认为是瞬间完成。给出所有的站名（最多包含 300000 个站）、所有线路（不超过 100000 条）的名称以及每条线路经过的站点名称，然后给出 Johny 和他朋友家附近的地铁站名称。

所有的名称只包含字母、数字、连字符（-）、单引号以及&。所有线路都是双向的，但是改变方向也算一次换乘，并且线路不会自交。

计算乘坐距离尽量长并且换乘次数最少的路线（输出格式详见原题样例）。

ACM/ICPC Europe - Northwestern

节省零钱（Cent Savings, Europe - Northwestern 2014, LA6952）

在超市买了 n（$1 \leqslant n \leqslant 2000$）件商品，其中第 i 个价格是 p_i（$1 \leqslant p_i \leqslant 10000$）分，结账时发现超市以分为单位对总价进行四舍五入。例如，94 分舍成 90，而 95 分就入到 100。

还可以使用 d（$1 \leqslant d \leqslant 20$）个分隔栏，插入到已经排成一排的 n 件商品中间，分成 $d+1$ 组单独结算，每组都可以四舍五入。现在需要计算如何插入分隔栏才能让总价最低。

ACM/ICPC South Pacific

最小公倍数（Least Common Multiple, South Pacific 2014, LA6783）

记 S 为一个正整数集，里面的数字表示成二十六进制时只包含 0 和 1。S 的前几个元素（十进制表示）分别是 1, 26, 27, 676, 677……

给出 3 个数，a、b 和 c，找出在 S 中，大于给定值 x 且刚好是 2^a、3^b、5^c 这 3 个数的最小的公倍数。输出其二十六进制表示。其中（$0 \leqslant a < 50$，$0 \leqslant b \leqslant 3$，$0 \leqslant c \leqslant 2$，$0 \leqslant x < 2^{63}$）。无解则输出“No Solution”。

迷信袜子（Superstitious Socks, South Pacific 2014, LA6789）

Trudy 非常迷信，她有 n（$2 \leqslant n \leqslant 100000$）只不同长度的袜子。只要穿过其中一双，就再也不穿那一双。经过 $\binom{2}{n}$ 天之后，就扔掉 n 只袜子再买。为了确保这一点，每天她就查看没穿过的每一双袜子，并且挑选两只长度最接近的那一双；如果有多双备选，那么选择其中那只更短的那一双。例如，她有 5 只长度分别为 1、2、4、6、15 的袜子，前 5 双袜子如表 4.3 所示。

表 4.3

第几天	1	2	3	4	5
袜子	(1, 2)	(2, 4)	(4, 6)	(1, 4)	(2, 6)
长度差距	1	2	2	3	4

给出每只袜子的长度，计算在第 k（$1 \leqslant k \leqslant 100000$ 且 $k \leqslant \binom{2}{n}$）天 Trudy 该穿哪一双？

ACM/ICPC Asia – Tokyo（东京赛区）

漂亮的空格（Beautiful Spacing, Asia - Tokyo 2012, LA6190）

给出一个包含 N（$2 \leqslant N \leqslant 50000$）个单词的文本，按照如下规则放入宽度为 W（$3 \leqslant W \leqslant 80000$）的行数无限的网格中，每个字符放一个格子。

（1）单词的顺序不能被打乱。

（2）同一行的两个单词之间至少得有 1 个空格。

（3）同一个单词的所有字符必须还是连接起来的。单词也不能被打破换行。

（4）每一行左边都不能留空格，除了最后一行外，右边也不可以留空格。

问如何布局使单词之间最大的空格最小。其中第 i 个单词的长度为 x_i（$1 \leqslant x_i \leqslant (W-1)/2$），$x_i$ 的长度上限保证了一定有一个布局能否符合上述规则。输出连续空格最大长度的最小值。

太空高尔夫（Space Golf, Asia - Tokyo 2014, LA6835）

要从水平面上发射子弹，到达指定目标。发射点和目标之间有一些垂直的阻碍。因为需要发射的初速度尽量低，所以最理想的方式是一路反弹的方式跃过这些阻碍。但是同时反弹的次数有限制。

图 4.32 给出了样例输入 4 中数据对应情况的一个粗略示意图。

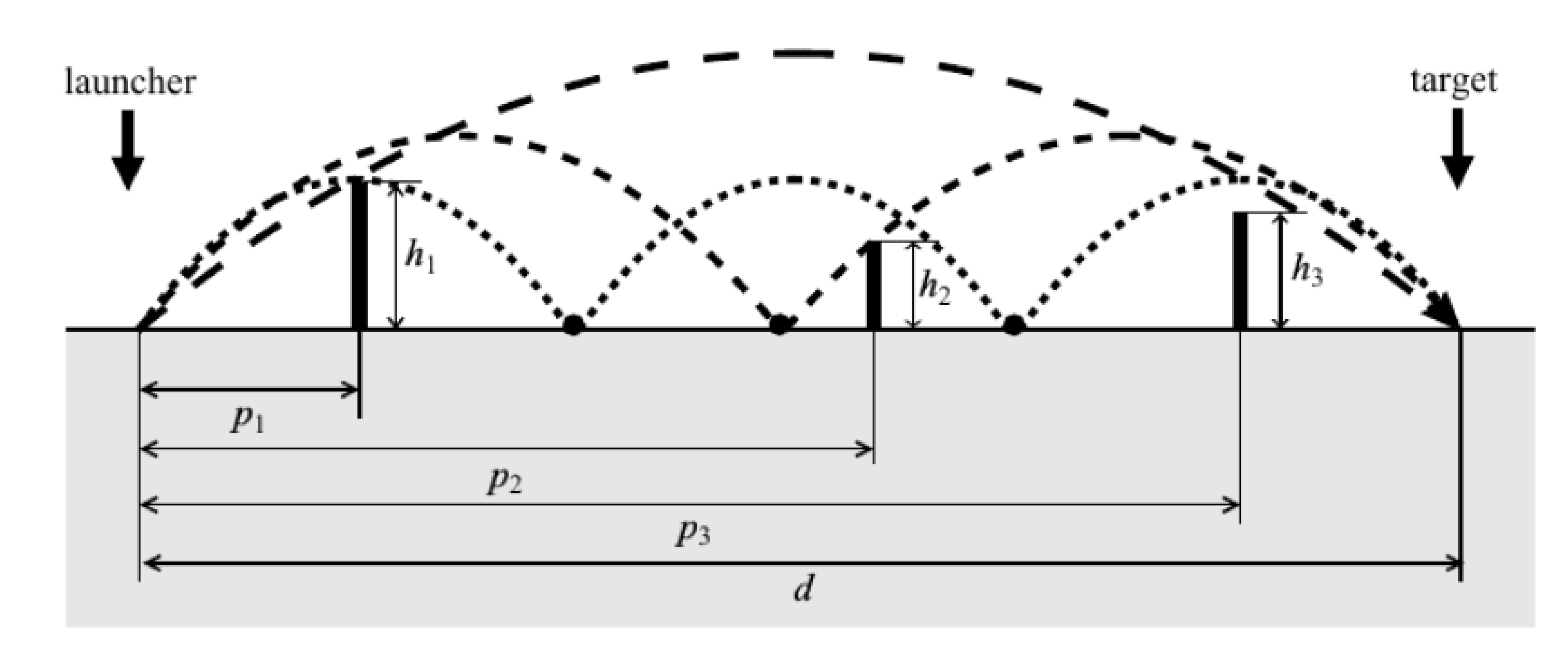


图 4.32

在本题中可以做如下假设：

（1）空气阻力可以忽略。

（2）水平面和子弹之间都是理想的刚性反弹。

（3）子弹可以认为是大小为 0 的质点。

（4）子弹发射瞬间的高度可以认为是 0。

假设子弹速度是 v，水平分量是 v_x，垂直分量是 v_y。初始时二者分别是 v_{ix} 和 v_{iy}。记 t 时刻子弹离初始点的水平距离为 x，垂直高度为 y。因为无空气阻力，所以有：$x = v_{ix}t$。假设重力加速度为 g，有 $y=-\dfrac{1}{2}gt^2+v_{iy}t$，推理过程参见原题。

由以上公式可得，子弹会在 $t=2v_{iy}/g$ 时接触地面。所以反弹点距离初始点的距离是 $2v_{ix}v_{iy}/g$。如果希望子弹飞出 l 的距离，子弹初速的两个分量需要满足 $2v_{ix}v_{iy} = lg$。

根据以上公式，可以得出子弹的抛物线轨迹方程：$y=-\left(\dfrac{g}{2v_{ix}^2}\right)x^2+\left(\dfrac{v_{iy}}{v_{ix}}\right)x$。

为简单起见，本题中重力加速度 $g = 1.0$。

给出初始发射点距离目标点的距离 d（$1 \leqslant d \leqslant 10000$），障碍物的个数 n（$1 \leqslant n \leqslant 10$），每个障碍物的位置 p_i（$pi<d$）和高度 h_i（$1 \leqslant h_i \leqslant 10000$），最大反弹次数 b（$0 \leqslant b \leqslant 15$）。输出能够让子弹到达目标点的初始速度的最小值 v_i。输出精确到小数点后 5 位，误差不超过 0.0001。

ACM/ICPC Asia – Aizu（爱知赛区）

隐藏的树（Hidden Tree Regionals, Asia - Aizu 2013, LA6669）

二叉树上每个叶子结点都赋有一个整数作为权值。如果每个非叶子结点的左右子树的权值之和相同，那么这棵树就认为是平衡的，图 4.33 就是一个例子。

如果一个平衡树从左到右把所有叶子结点的权值全部列出来形成的序列是序列 A 的子序列（不一定连续），那么就说这棵树隐藏在 A 中。图 4.33 中的树就隐藏在序列 3 4 1 3 1 2 4 4 6 中，因为 4 1 1 2 4 4 是其子序列。

给出一个长度为 N（$1\leqslant N\leqslant 1000$）的整数序列 $A_1A_2\cdots A_N$（$1\leqslant A_i\leqslant 500$），找出隐藏在其中的叶子个数最多的平衡树并输出其叶子个数。图 4.33 中的树就是上述序列中隐藏的叶子个数最多的平衡树。

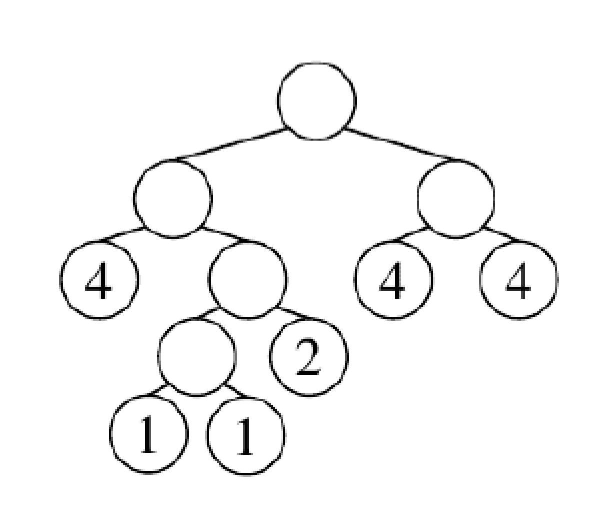


图 4.33

ACM/ICPC Asia – Fukuoka（福冈赛区）

城市合并（City Merger, Asia - Fukuoka 2011, LA5856）

有 n（$n\leqslant 14$）个城市要合并，但是新的城市名称必须包含以前每一个城市名称（长度≤20 且唯一）作为连续子串，计算满足上述条件的新城市名称的最短长度。

海盗的宝藏（Captain Q's Treasure, Asia - Fukuoka 2011, LA5857）

现在有一个岛的一张旧地图，里面标示了岛上的宝藏埋藏的位置。

地图可以认为是方块组成的矩形网格，每个方块可能有个数字（也可能没有），数字标示了这个格子和 8 个相邻方格组成的 9 宫格的宝藏数量。每个方块里面最多有 1 个宝藏，但是整个地图中的宝藏数量是未知的。需要计算出岛上埋藏的宝藏数量的最小值。

地图的宽高为 w,h（$1\leqslant h,w\leqslant 15$）。输入一个 $w*h$ 的字符方阵，每一个字符对应地图上一个方块区域，“.”表示是海水，没有宝藏。“*”表示岛屿上一个宝藏数量位置的方块。0～9 表示以此为中心的 9 宫格中的宝藏数量。

往返旅程（Round Trip, Asia - Fukuoka 2011, LA5860）

Jim 计划去山区访问他的好朋友。行程包含去程和回程。需要计算出来回的最小代价。

出发地所在的镇（编号为 1）和目的地所在镇（编号为 n）之间是一个路网，连接 n 个镇（$2\leqslant n\leqslant 50$），包含 m 条路（$0\leqslant m\leqslant n*(n-1)$）。每条路都是单向的，不会有重边，第 j 条路的费用是 c_j（$1\leqslant c_j\leqslant 1000$）。通过一个镇 i 时也需要付费 d_i（$1\leqslant d_i\leqslant 1000$），但同一个镇如果经过多次只需一次付费。

每个镇都有海拔，第 i 个镇的海拔 e_i 满足（$1 \leqslant e_i \leqslant 999$）。在去程路上，如果经过 a 到 b，那么 a 的海拔必须不大于 b。回程时如果经过 a 到 b，那么 a 的海拔必须不小于 b。计算出整个旅程所需要的最少费用，包含经过每个镇子的费用。问题无解则直接输出-1。

ACM/ICPC Asia – Tehran（德黑兰）

旅馆（Hotel, Asia - Tehran 2005, LA3550）

有一个导游要为她的旅客订旅馆房间，房间的床位数以及价格都不同。希望每个房间住尽量多的人，但是面临一些限制：

（1）异性且不是夫妻的二人不能住一个房间。

（2）如果房间里面已经有夫妻，就不能再住人了。

（3）夫妻不是必须安排到同一间。

（4）也可以有些房间未住满。

给出旅客中的男性人数 m（$0 \leqslant m \leqslant 500$），女性人数 f（$0 \leqslant f \leqslant 500$），以及已经预定的房间数目 r（$0 \leqslant r \leqslant 500$），其中总共有多少对夫妻 c（$c \geqslant 0$）。注意不会有重婚：每个人最多有一个伴侣。

对于 r 个房间，给出每个房间的床位数 b（$1 \leqslant b \leqslant 5$）和价格 p（$1 \leqslant p \leqslant 1000$）。输出将所有的旅客安排好所需的最小的总价格。

拦截导弹（Intercepting Missiles, Asia - Tehran 2005, LA3551）

有很多轰炸机来轰炸首都，要用导弹打下来。但是还有许多客机不能被打中，为简化问题可做如下假设：

（1）所有的导弹都是在水平面上，位置固定，发射之后垂直向上飞，x 坐标不变。

（2）整个天空可以认为是一个二维平面，所有飞机沿着和 x 轴平行的方向从左往右飞，y 坐标不变。而且所有飞机的 y 坐标不重复。

（3）从时间点 0 开始，可以发射每个导弹。

（4）每架飞机都是同样的单位长度，但是导弹没有长度。

（5）只要导弹击中飞机或者接触到其边缘，就会爆炸。同时被击中的飞机会在爆炸之前继续正常飞行一段时间。可以假设被击中的轰炸机会在飞越所有的防空导弹之后再爆炸。在此之前，仍然可被其他导弹击中。

输入 m,n,k（$0 \leqslant m, n, k \leqslant 300$），分别是轰炸机、客机和导弹的数量。给出三者的速度 v_m,v_n,v_k（$1 \leqslant v_m, v_n, v_k \leqslant 10000$）。给出在时间点 0 所有飞机的 x,y 坐标。再给出每个导弹的坐标。所有的坐标值都是不大于 10000 的非负整数。

输出在不击中客机的前提下，能被击中的轰炸机个数的最大值。

钻石（Diamonds, Asia - Tehran 2009, LA4649）

国王命令大臣给王子设计一个游戏。其中有 N（$1 \leqslant N \leqslant 500$）个标记为 $1 \sim N$ 的盒子，



每个盒子上有一个只能由一把唯一钥匙打开的锁。游戏开始之前，大臣把每个盒子 i 的钥匙复制了两份，分别放在盒子 $i+1$ 和 $i-1$ 中（只要这两个编号的盒子存在）。然后把 D（$0 \leqslant D \leqslant N$）个钻石放到 D 个不同的盒子中。最后所有的盒子都锁上，并且给了王子其中 M（$1 \leqslant M \leqslant N$）个盒子的钥匙。王子需要打开尽量少的盒子来收集所有的钻石。

给出所有放钻石的 D 个盒子的编号，以及给了钥匙的 M 个盒子的编号。输出要收集到所有钻石所必须打开的盒子的个数的最小值。

机器人跟踪（Tracking Robots Asia - Tehran 2009, LA4654）

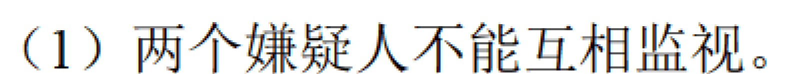
有一些机器人在一个区域内移动，并且把位置发送给服务器。这个区域是一个封闭的多边形，分成编号为 1～N（$1 \leqslant N \leqslant 100$）的 N 个互不重叠的区域，如图 4.34 所示。一开始机器人都在区域 1 内，然后开始移动。当机器人进入一个新区域，就把位置发给服务器。注意机器人可以在一个区域内随时任意进出。

服务器只是收到长度为 M（$1 \leqslant M \leqslant 200$）的区域编号序列，但是不知道具体发送机器人的编号。给出区域的相邻关系，计算出机器人数量可能的最大和最小值。

图 4.34

叛徒（Traitor, Asia - Tehran 2013, LA6745）

有一个安全组织，有编号为 1 到 n 的 n（$1 \leqslant n \leqslant 10000$）个特工，内部是一个等级结构（如图 4.35 所示）。每个特工都有一个领导，并且也有一些顶级领导不被任何人领导。现在怀疑组织中有叛徒，有 k（$1 \leqslant k \leqslant n$）个嫌疑人，需安排人互相监视。安排规则如下：



（1）两个嫌疑人不能互相监视。

（2）嫌疑人只能被他的领导或者直接下属监视。

（3）任何人都不能监视超过 1 个嫌疑人。

图 4.35

给出每个特工的领导编号（0 是顶级领导），以及 k 个嫌疑人编号。输出按照以上的监视规则最多能安排多少嫌疑人能同时被监视。

天线（Antennas, Asia - Tehran 2014, LA7016）

在一维世界，有 n 间房子（$0<n \leqslant 5000$），每个都有坐落区间$[a,b]$（$0<a \leqslant b<10^9$）。房内都有 SIM 卡，卡分 1 型和 2 型。现需要选择位置安装天线，若天线安装在点 x，那么其覆盖区间就是$[x-R, x+R]$（$0<R \leqslant 10^9$），只要房子所在区间内有 1 个点在此覆盖区间，就认为被这个天线覆盖。天线有 3 种类型：1 型可覆盖 1 型 SIM 卡，2 型可覆盖 2 型 SIM 卡，通用型可覆盖两种 SIM 卡。费用分别为 C1,C2,C3（(max(C1, C2)<C3<C1+C2)），3 种天线的覆盖范围 R 都是一样的，所有的费用都不大于 10^9。

给出房间数据以及 SIM 卡类型，计算如何布局天线才能覆盖所有房间，并且总费用最小。输出其最小值。

龙之国度（Dragons, Asia - Tehran 2014, LA7018）

龙之国度里有编号为 1～N 的 N 个城市，以及 M 条仅连接两个城市的道路（$1 \leqslant N \leqslant 300$，

$0 \leqslant M \leqslant N(N-1)/2$）。有 K（$1 \leqslant K \leqslant 1000$）条龙 $D_1, D_2, \cdots, D_K$ 住在这些城市里。其中龙 D_i 生活在城市 C_i，一开始有 S_i 个头，只要活着，就每分钟长出 N_i 个新头（$1 \leqslant C_i \leqslant N$，$1 \leqslant S_i \leqslant 10^5$，$0 \leqslant N_i \leqslant 10^5$）。只要还有一个头在，龙就依然存活。

现在需要一些勇士去杀掉所有的龙。一开始给每个勇士安排一城市。每分钟，勇士要么走向相邻的城市，或就地选择一个活龙砍掉一个头。每分钟，都可指挥每个勇士的行动，当这一分钟结束时，每个活着的龙 D_i 就长出 N_i 个新头。

计算最少需要多少个勇士才能保证可在有限的时间内杀掉所有的龙。

过路费（Toll, Asia - Tehran 2014, LA7019）

丝绸之路如今被强盗占据，经过的区域被划分成 n（$n \leqslant 1000$）个正方形领地（之间可能有重叠），作为不同强盗的地盘。每个过路费都是 1，其收取规则如下：

（1）强盗不能在其地盘外收过路费。

（2）每个强盗在收到路过他地盘商人的过路费之后，必须发一个通行证。在同一个地盘内，其他强盗不能再向这个商人要钱。一旦这个商人走出通行证对应的地盘，通行证自动失效。

（3）若无有效的通行证，商人不能经过任何强盗地盘。

（4）如果商人愿意，他可以自己把通行证弄失效，并且从任一个地盘覆盖所在位置的强盗那里买一个新通行证。

为简化起见，丝绸之路可以认为都是水平或者垂直的。

现在按照在道路上出现的顺序给出丝绸之路的道路的必经结点的个数 m（$m \leqslant 1000$）以及每个结点坐标。再给出每个领地的左下角顶点坐标（x,y）以及边长 k（$x, y \leqslant 10^6$，$k \leqslant 1000$）。计算要把丝绸之路的道路从头到尾走下来，至少要交多少过路费。

班车（Bus, Asia - Tehran 2014, LA7021）

公司为 n 个员工提供连续 d 天的班车服务，每天都要付给司机 p（$n,d \leqslant 500$，$p \leqslant 10^9$）元。每天在车上选择一人给司机付款。记第 i 天乘车的人数是 n_i，对于员工 A 来说，假设他分别在第 $t_1, t_2, \cdots, t_k$ 天乘了 k 天的车。那么他应该承担的合理费用就是：$P_A = p*(1/n_{t1} + 1/n_{t2} + \cdots + 1/n_{tk})$。如果 A 被 r 次选中付款，那么他就多付了 $E_A = r*p - P_A$。对于一个付款方案来说，其公平度定义为所有 E_A 的最大值。所有方案中公平度最小的那个就称为公平方案。计算公平方案的公平度。

ACM/ICPC Asia – Daejeon（韩国大田）

武器装备（Equipment, Asia - Daejeon 2011, LA5842）

有 N（$1 \leqslant N \leqslant 10000$）个编号为 $1 \cdots N$ 的武器，每个武器 R 的评分是一个 5 维向量$\{r_1, r_2, r_3, r_4, r_5\}$。都是 0～10000 之间的整数。

如果一个士兵选择了多个武器，那么他每一维相应的能力都会得到加成，但是加成值



不是多个武器对应向量评分的和，而是其最大值。定义目标评分为一个士兵选择了 K 个兵器之后，各个维度的能力加成值的总和。

从 N 个武器中选择 K（$1\leqslant K\leqslant N$）个，求出目标评分最大值。

倒水（Buckets, Asia - Daejeon 2013, LA6501）

给出两个水桶 A 和 B，容量分别为 a、b（升）。一开始分别有 x 和 y 升水，其中（$1\leqslant a,b,x,y\leqslant 10000000000\leqslant x\leqslant a$，$0\leqslant y\leqslant b$）。可以对两桶水进行如下操作：

- 把 A 或 B 中水全部倒掉。
- 把 A 或 B 装满水。
- 把 A 的水往 B 中倒，直到 A 变空或者 B 变满。
- 把 B 的水往 A 中倒，直到 B 变空或者 A 变满。

定义整数对 $O_i=(s_i,t_i)$为一个目标容量。如果存在 0 或多个上述操作形成的序列，可以将水量分别为 s 和 t 升的两个桶变成(s_i,t_i)，则称这个目标容量从(s,t)可达。

给出一个目标容量序列 $O_1, O_2, O_3, \cdots, O_n$，其中 $1\leqslant n\leqslant 200000$。需要找到一个最长的子序列 $O_{i_1}, O_{i_2}, O_{i_3}, \cdots, O_{i_l}$（其中 $i_1<i_2<\cdots<i_l$ 且不必连续），使得每一个都从前一个可达。O_{i_1} 的前一个是(x,y)。输出这个子序列的长度。

高尔夫球场（Golf Field, Asia - Daejeon 2013, LA6503）

给出一个平面，有 n（$4\leqslant n\leqslant 30000$）个点，坐标为 $x_1\, y_1\, x_2\, y_2\, \cdots\, x_n\, y_n$，其中（$-1000000000\leqslant x_i, y_i\leqslant 1000000000$），输入保证任意 3 点都不共线。需要选择 4 个点，使得其凸包面积最大。给定点集的凸包指的是把这些点包在边上或者内部的凸多边形中最小的那个。

在图 4.36（a）中，如果选择 1,3,5,8 这 4 个点，则以这 4 个点为顶点四边形都不是凸的。所以这 4 个点的凸包就是以 1,3,8 为顶点的三角形。另外，不难看出这个三角形就是所有 4 个点的凸包中面积最大的。图 4.36（b）中最大的凸包就是顶点为 1,3,5,8 的四边形。

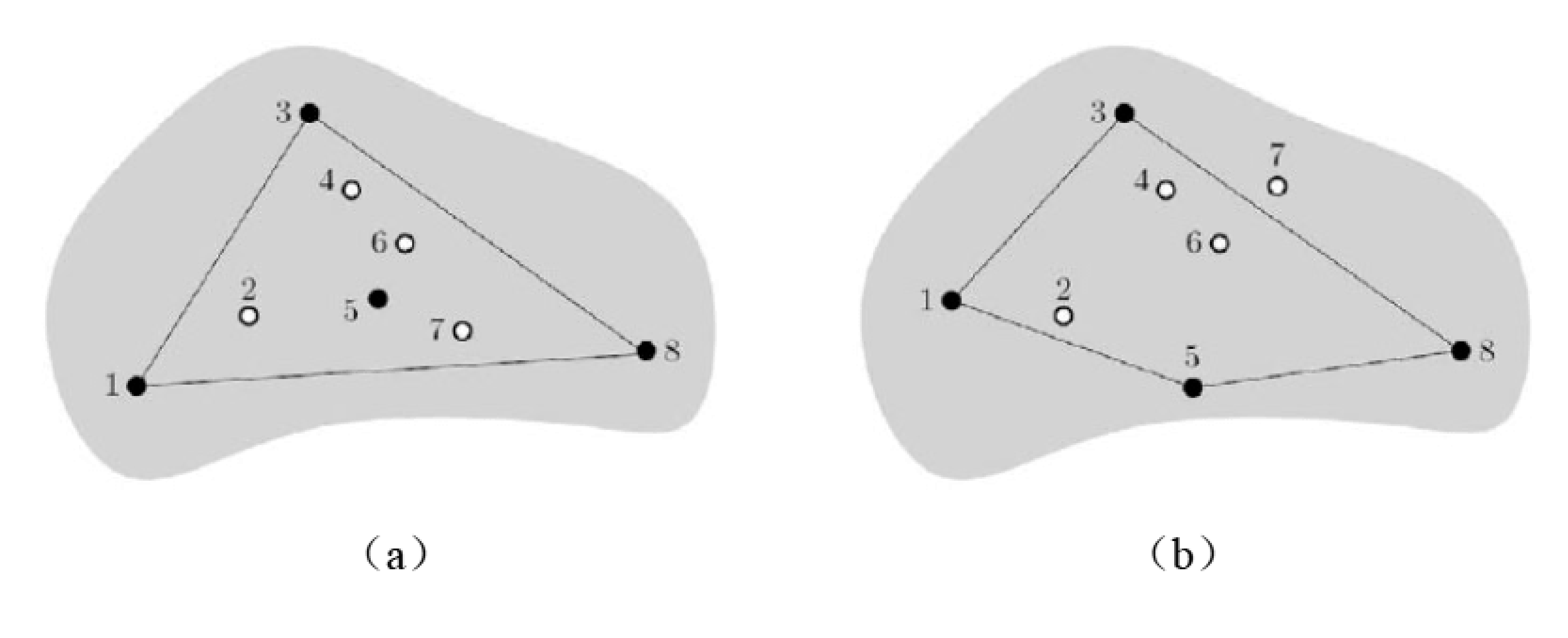
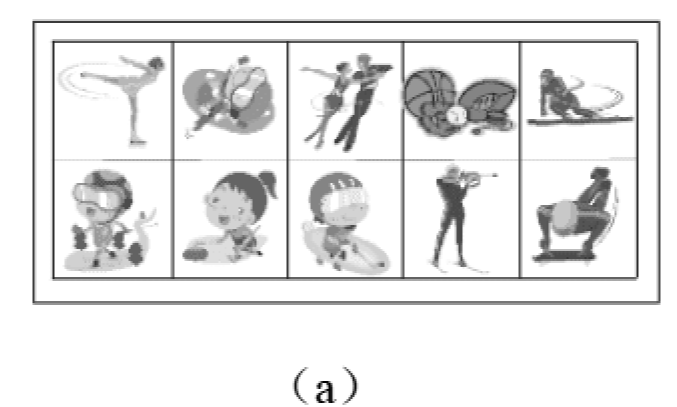


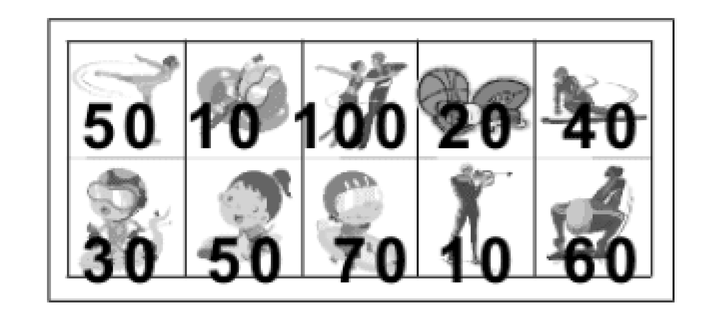
图 4.36

贴纸（Stickers, Asia - Daejeon 2013, LA6510）

有一个贴纸，里面分成两排各 n（$1\leqslant n\leqslant 100000$）块（如图 4.37 所示），每块给出其撕下之后的价值 p（$0\leqslant p\leqslant 100$），如果撕掉其中一块就会将 4 个相邻的贴纸其中的一个弄坏。所以需要选择一个价值和最大的子集来撕，并且其中的任意两块都不相邻（对角不算相邻）。计算能选择出来的最大的价值和。



（a）



（b）

图 4.37

字符串转换（String Transformation, Asia - Daejeon 2014, LA6901）

字符串 S 如果符合以下规则，那么就称它是合法的。

- S = 'ab'。
- S = 'aTb'，其中 T 是合法的。
- S = TT，其中 T 是合法的。

例如，'aabbabab'、'abababab' 和 'aaaabbbb'都是合法的。

对于两个合法字符串 A 和 B，要把 A 通过一系列的交换相邻字符的操作转换成 B。例如，A=aabbabab，B= aaaabbbb 就可以做如下的转换：

aabbabab → aabbaabb → aababab → aabaabbb → aaababbb → aaaabbbb

给出 A 和 B（长度都小于 100000），计算出要把 A 转换成 B，最少需要多少次转换操作。注意每次操作之后的结果必须合法。

3 个正方形（Three Squares, Asia - Daejeon 2014, LA6902）

一个正方形，如果所有的边都和坐标轴平行，就称它为轴平行的。两个正方形边长相等就是全等的。

给出一个包含 n（$1 \leqslant n \leqslant 100000$）个点的点集 P，对于实数 $x \geqslant 0$，如果存在 3 个边长为 x 的全等轴平行正方形能够覆盖 P 中的所有点，那么就称 x 对 P 是 3SQ−充分。如果边长为 0，正方形可认为缩成了一个点。点在正方形的边上也认为被覆盖。

对于指定的 P，找出对其 3SQ−充分的最小 x。P 中每个点的两个坐标都在闭区间 [−1000000000, 1000000000]内。

交通卡（Travel Card, Asia - Daejeon 2014, LA6904）

城市中有公交和轨道交通，同时发行多种公共交通卡，价格规则如下：

（1）每次公交 1 元，轨道 2 元。

（2）每次换乘都要重新买票。

（3）单天公交卡 3 元，一天内可乘任意多次公交，坐轨道要钱。

（4）单天旅行卡 6 元，一天内可乘任意多次公交和轨道。

（5）7 天公交卡 18 元，7 天内可乘任意多次公交，坐轨道要钱。

（6）7 天旅行卡 36 元，7 天内可乘任意多次公交和轨道。

（7）30 天公交卡 45 元，30 天内可乘任意多次公交，但是坐轨道要钱。

（8）30 天旅行卡 90 元，30 天内可乘任意多次公交和轨道。

现在给出 n（$1 \leqslant n \leqslant 10000$）天的旅行计划，每天都给出乘公交次数 a 和乘轨道的次数 b（$0 \leqslant a, b \leqslant 100000$）。计算出整个计划所需的最小费用。

两个游艇（Two Yachts, Asia - Daejeon 2014, LA6905）

ICPC 的海军将要拍卖两个豪华游艇在下一季的使用权。每一个竞拍者必须提供一份正式的竞拍标书，里面包含以天为单位的使用时段以及竞拍价格。假设下一个季度包含编号为 1～m 的 m 天。

现在有 n（$1 \leqslant n \leqslant 10000$）份标书。每个都包含使用起始时间点第 s 天，结束点 t，以及竞拍价 p。其中，$1 \leqslant s \leqslant t \leqslant 10000000$ 并且 $1 \leqslant p \leqslant 100000$。现在需要从 n 份标书中选择一个子集，使得其竞拍价之和最大。前提是选中的任意两份标书中的时间段不能相互重叠。

ACM/ICPC Asia – Harbin（哈尔滨赛区）

Alice 和 Bob 的旅游（Alice and Bob's Trip, Asia - Harbin 2010, LA5088）

Alice 和 Bob 一起出去旅游，城中有编号为 0～n−1 的 n（$1 \leqslant n \leqslant 500000$）个景点，所有的景点由有向边连接，形成一棵树。他们从根结点 0 出发，轮流选择要走的下一条边。Bob 先开始选，要满足总距离在区间[L,R]（$0 \leqslant L,R \leqslant 1000000000$）内。Bob 总是选使总距离最大的路径走，Alice 总是选使总距离最小的路径走，而且他们都会选择最优策略，求最后走出的总距离的最大值。

输入景点的个数 n，每条边的连接的结点 a,b 以及长度 c（$1 \leqslant c \leqslant 1000$）。求走出的总距离，如果总距离在[$L$,$R$]的范围外，输出“Oh, my god!”。

ACM/ICPC Asia – Changchun（长春赛区）

土豪（LA7183, Asia - Changchun 2015, Too Rich）

你手上有各种货币，给出每种面值对应的数量：$c_1, c_5, c_{10}, c_{20}, c_{50}, c_{100}, c_{200}, c_{500}, c_{1000}, c_{2000}$（$0 \leqslant c_i \leqslant 100000$）。要购买一个价格 p（$0 \leqslant p \leqslant 10^9$）的物品，希望用各种面值的货币组合支付，且要求货币的数量最大，输出货币数量的最大值。如果问题无解，输出−1。

例如，p=17，你有两个 10 元，4 个 5 元，8 个 1 元。就使用 2 个 5 元和 7 个 1 元支付，问题的解就是 9。

废墟重建（Rebuild, Asia - Changchun 2015, LA7187）

水平面上有 n 个点$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$形成一个封闭的路径。对于 $1<i \leqslant n$，点 i 和 i−1 相邻，1 和 n 也相邻。任意相邻两点之间的距离都是正整数。

现在需要在每个点上画圆，其中第 i 个点上的圆半径为 r_i。要求相邻两点对应的圆互相外切。如果两点不相邻，对应的圆之间没有任何限制。r_i 可以为 0，但是不能为负数。

输入 n 和每一个 x_i,y_i（$3\leqslant n\leqslant 10^4$，$|x_i|,|y_i|\leqslant 10^4$），确定所有的 $r_1, r_2, \cdots, r_n$，使得所有圆的面积之和最小。输出最小的总面积以及每一个 r_i。如果问题有多解，输出任意一组。如果问题无解，输出“IMPOSSIBLE”。

举例来说，对于点(0,0), (11,0), (27,12), (5,12), 可以选择 (r_1,r_2,r_3,r_4)=(3.75,7.25,12.75,9.25)。那么总的面积就是 $3.75^2\pi+7.25^2\pi+12.75^2\pi+9.25^2\pi = 988.816$。注意重叠的部分也要计算两次，如图 4.38 所示。

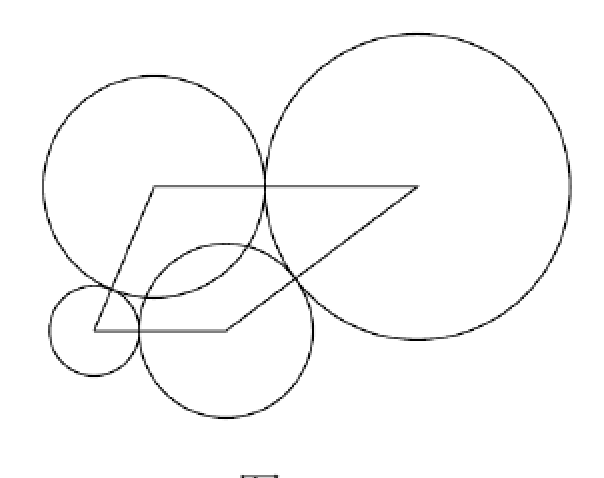


图 4.38

部分树（Partial Tree, Asia - Changchun 2015, LA7190）

给出一个函数 $f(d)$，以及 $f(1),f(2),\cdots,f(n-1)$的值，其中 $0\leqslant f(i)\leqslant 10000$。给出 n（$2\leqslant n\leqslant 2015$），对于 n 个结点的树，有 n^{n-2} 种构造方式。对于其中一种构造方式，定义一个结点的 C 值为 $f(d)$，其中 d 是这个结点的度数。计算树的所有构造方式中所有结点 C 值之和的最大值。

输入案例中，最多有不超过 10 个 $n>100$。

棋盘解密（Chess Puzzle, Asia - Changchun 2015, LA7191)

给出一个 n 行×m 列（$1\leqslant n,m\leqslant 100$）的棋盘。坐标$(i,j)$（$1\leqslant i\leqslant n$，$1\leqslant j\leqslant m$）对应第 i 行第 j 列的方格，有黑白两色的两种棋子。初始某些格子中已有棋子，其他是空的。需要选用棋子将空格子放满，黑白棋子都是无限量供应的。给出两个整数 a,b（$1\leqslant a\leqslant n$，$1\leqslant b\leqslant m$），对于一个特定的放置方案，按照如下方案计算其得分：

对于任意两个格子(x_1, y_1)和(x_2,y_2)，如果满足以下 3 个条件，就得 1 分：

（1）(x_1, y_1)中有黑棋子。

（2）(x_2,y_2)中是白棋子。

（3）$|x_1 - x_2| = a$ 且$|y_1 - y_2| = b$。

计算总分最小的棋子放置方案，然后输出这个方案中每个格子的棋子颜色（W 白色，B 黑色）。如果有多种方案，输出字典序（其中 B<W）最小的方案。详细的输入输出格式请参考原题。

ACM/ICPC Asia – Shenyang（沈阳赛区）

宝塔（Pagodas, Asia – Shenyang 2015, LA7241）

有标记为 1～n 的 n（$2\leqslant n\leqslant 20000$）个佛塔排成一列。现在只有 a,b 两个依然完好。两个和尚 Yuwgna 和 Iaka 决定轮流重建其他佛塔，Y 先手。每一轮可以选择重建佛塔 i（$i\notin\{a,b\}$，$1\leqslant i\leqslant n$），前提是佛塔 j 和 k 完好或已经建好，且 $i=j+k$ 或 $i=j-k$。每个编号只能重建一次。这等于是两个和尚之间的游戏，轮到谁无法新建佛塔就算输掉。

假设两人每一步都是最优策略，输出最后胜者的名字。

青蛙（Frogs, Asia – Shenyang 2015, LA7243）

编号为 0～m−1 的 m 个石头排成一个圈，有编号为 1～n 的 n 个青蛙在上面跳（$1 \leqslant n \leqslant 10^4$，$1 \leqslant m \leqslant 10^9$）。第 i 个青蛙每一步可以正好跳过 a_i 个石头（$1 \leqslant a_i \leqslant 10^9$），意思是可以从石头 $j \bmod m$ 跳到 $(j+a_i) \bmod m$。

所有的青蛙从石头 0 开始一直跳。一个青蛙在到达一个石头时会占有它，并且会持续地跳跃来占有更多的石头。即使青蛙跳开了，这个石头也认为是被占有过了。输出能被占有至少一次的那些石头的编号之和。

飞行马戏团（Game of Flying Circus, Asia – Shenyang 2015, LA7244）

一个边长为 300m 的正方形场地，4 个角按照顺时针方向放置有编号依次为 1,2,3,4 的浮标。场地上举行飞行比赛。选手开始时都从#1 出发，他们要按照顺时针方向依次触碰 4 个浮标(2,3,4,1)。可以在场地边缘或者场地内部自由地飞。

两种情况下，玩家可得 1 分：

（1）在对手之前触碰了一个浮标。如果你的对手在你之前碰到浮标 2，他就得 1 分。而且必须严格遵从上文给出的顺序来触碰浮标。

（2）打架得分，如果你和对手在一个点遇到，你可以跟他打一架，胜者得 1 分。为了游戏的平衡，在有人碰到浮标 2 之前，不允许打架。

有 3 种类型的玩家：

- ❑ 速度型：专注于速度，在通过触碰游标得分的同时。尽量避免打架。
- ❑ 战斗型：专注于打架，尽量通过打架得分，比速度型的慢，如果对手是速度型的，很难通过触碰浮标得分。
- ❑ 全能型：速度和战斗兼顾。

现在有个全能型的选手 Asuka 和速度型的 Shion。二人比试，并且规则简化：谁先碰到浮标 1 就算结束。Shion 的策略很简单：使用最短路径依次触碰 2,3,4,1。

Asuka 擅长打架，只要和 Shion 打架就能得 1 分，并且对手会在此后晕倒 T($0 \leqslant T \leqslant 2000$)秒不能动。但她比 Shion 慢，所以希望在比赛过程中跟 Shion 打一架。而且如果二人同时触碰浮标，这一分会判给 Asuka，并且二人一定会马上打一架。

Asuka 的速度是 V_1m/s，Shion 的速度是 V_2m/s（$0 \leqslant V_1 \leqslant V_2 \leqslant 2000$）。计算 Asuka 有可能通过更多的得分获胜吗？如果有可能就输出“Yes”，否则输出“No”。

样例输入：

```
2
1 10 13
100 10 13
```

样例输出：

```
Case #1: No
Case #2: Yes
```

提示：

在案例 2 中，Asuka 可以飞到浮标 2 和 3 的中点，并且等待 Shion 到达之后打一架，然后 Shion 晕倒 100 秒不能动。接着 Asuka 可以飞回浮标 2，但是不能得分。但是之后她可以依次飞到 3,4,1 并且得 3 分。

棋盘（Chessboard, Asia – Shenyang 2015, LA7245）

有一个 n 行×m 列的棋盘。从上到下的每一行编号依次为 1～n，从左到右每一列编号为 1～m。棋盘上有 o 个损坏的格子不能放棋子。R 和 D 二人玩一个游戏。D 给 R 一个由 UDLR 4 个字符组成的长度为 l 的命令字符串，其中(x,y)代表棋子的当前位置，每个字符对应的命令如下：

- U：从 格子(x,y) 移到 $(x-1,y)$。
- D：从 (x,y)移到 $(x+1,y)$。
- L：从(x,y)移到$(x,y-1)$。
- R：从(x,y)移到$(x,y+1)$。

接着 R 就在棋盘的某个格子上放一个棋子，然后根据 D 给出的命令来移动这个棋子。如果会让棋子走出棋盘边界或进入损坏格子，R 就忽略这个命令并继续执行下一条。输入 o 个损坏的格子的位置，以及命令字符串。对于每个未损坏的格子(i,j)，假设从(i,j)开始的棋子会停在$(x(i,j),y(i,j))$，输出所有 $(i-x(i,j))^2+(j-y(i,j))^2$ 之和。数据范围：（$1\leqslant n,m,o,l\leqslant 1000$）。

Kykneion 哮喘（Kykneion asma, Asia – Shenyang 2015, LA7248）

给出整数 n（$2\leqslant n\leqslant 15000$）和 5 个整数 a_i（$0\leqslant i\leqslant 4$，$0\leqslant a_i\leqslant 30000$）。计算十进制表示长度为 n，包含不超过 a_i 位数字 i 且不包含 5,6,7,8,9 的数字的个数（不能包含前导 0）。输出其模 10^9+7 的值。

数字连接游戏（Number Link, Asia – Shenyang 2015, LA7249）

给出一个 n 行×m 列（$1\leqslant n,m\leqslant 50$）的网格，某些包含非零数字，其他为空。使用两种类型的路径：

（1）I 型连接两个奇偶不同的数字所在的格子。

（2）II 型是一种环形连接，其中唯一的一种特殊情况是连接两个相邻空格的路径（如图 4.39 左上角的线段）。

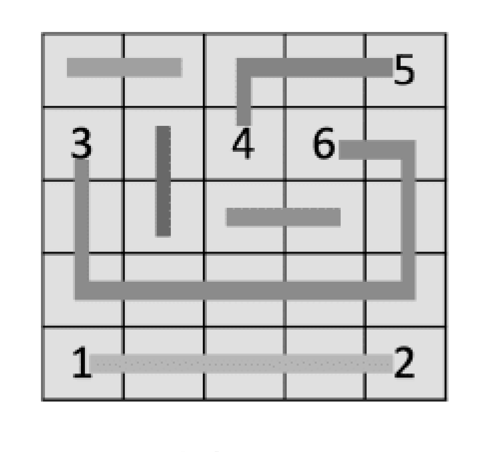


图 4.39

路径不能相交（占用同一个格子）。而且路径都是有费用的，从 (a,b) 连接到相邻的(c,d)。需要付出一定费用，反过来连接的费用相同。一条路径的总费用就是沿着路径从第 1 个格子走到最后一个格子所用的费用总和。在环形路径中，费用加倍。

游戏的总费用就是所有路径的费用之和。给出网格中每个格子上的数字，以及相邻格子之间的费用，计算能否有一种费用最小的连接方案使得每个格子刚好在一条路径上。如果问题有解，输出其最小费用。否则输出“-1”。详细的输入和输出格式请参考原题。所有的输入数字以及问题的解，都可用 32 位有符号整数表示。



样例输入：

```
3
3 3
1 0 0
1 0 0
2 0 2
1 2 1
2 1 1
3 1
5 6
1 4
1 4
1 1 2 2
1 2 3
3 5
0 0 0 0 0
0 5 0 6 0
0 0 0 0 0
1 1000 1000 1000 1
1 1000 1000 1000 1
1 1 1 1
1000 1 1 1000
1 1 1 1
```

样例输出：

```
Case #1: 10
Case #2: -1
Case #3: 14
```

提示：

图 4.40 就是输入样例 1 和 3 的解。对于红色路径，需要双倍费用，因为是环形路径。

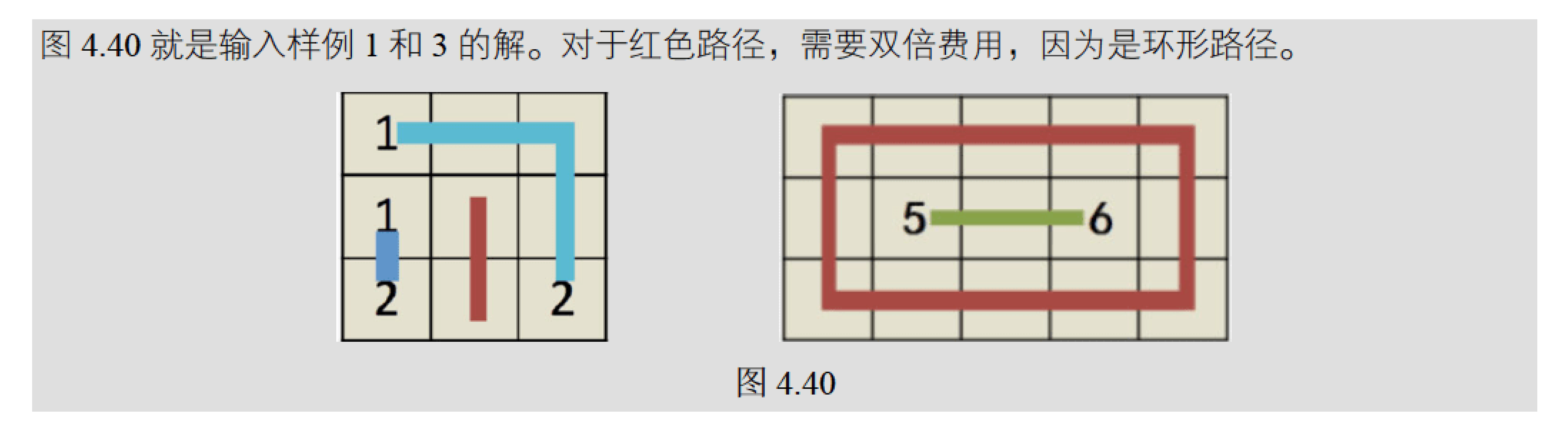


图 4.40

ACM/ICPC Asia – Dalian（大连赛区）最后的谜题（The Last Puzzle, Asia - Dalian 2011, LA5695）

英雄要打开一扇大门之前，必须让一排 n（$1 \leqslant n \leqslant 200$）个按钮全都变成按下状态。第 i 个按钮在按下 T_i（$1 \leqslant T_i \leqslant 1000000$）秒之后就会自动弹起来。从第 1 个到第 i 个按钮的距离是 D_i（$1 \leqslant D_i \leqslant 1000000$），也就是说，按完 1 之后需要 D_i 秒的时间来按 i，按完 D_i 之后也就需要 $|D_j - D_i|$ 的时间去按 D_j。每个按钮只能按一次。计算如果要让所有的按钮都处于按下状态，要以何种顺序按下这 n 个按钮？输出 n 个整数，表示依次按下的按钮编号，问题有多解则任意输出一个。问题无解则输出“Mission Impossible”。

提示：

在第二个输入样例中，无论先按哪个按钮，这个按钮都会在按第二个按钮之前弹起来，所以问题无解。

样例输入：

```
2
4 3
0 3
2
3 3
0 3
4
5 200 1 2
0 1 2 3
```

样例输出：

```
1 2
Mission Impossible
1 2 4 3
```

十六进制视图（Hexadecimal View, Asia - Dalian 2011, LA5696）

对于给定的数据，需要提供十六进制的视图。包含多行，除最后一行外，每一行对应 16 个字符。每一行包含以空格分隔开的 3 列。

（1）地址：4 位宽的以十六进制表示的起始地址。

（2）内存转储：这一行的十六进制表示，每两个字符之间有一个空格。如果最后一行不够 16 个字符，在后面补上空格。其中十六进制字母使用小写。

（3）文本：这一行的 ASCII 表示，大写字母要转成小写，小写转成大写。

给出一行文本，输出其十六进制表示。



样例输入：

```
Hex Dump
#include <cstdio>
printf("Hello, World!\n");
main = do getLine >>= print . sum . map read . words
```

样例输出：

```
0000: 4865 7820 4475 6d70                          hEX dUMP
0000: 2369 6e63 6c75 6465 203c 6373 7464 696f #INCLUDE <CSTDIO
0010: 3e                                            >
0000: 7072 696e 7466 2822 4865 6c6c 6f2c 2057 PRINTF("hELLO, w
0010: 6f72 6c64 215c 6e22 293b                ORLD!\N");
0000: 6d61 696e 203d 2064 6f20 6765 744c 696e MAIN = DO GETlIN
0010: 6520 3e3e 3d20 7072 696e 7420 2e20 7375 E >>= PRINT . SU
0020: 6d20 2e20 6d61 7020 7265 6164 202e 2077 M . MAP READ . W
0030: 6f72 6473                               ORDS
```

排列的签名（Number String, Asia - Dalian 2011, LA5697）

1～n 的某个排列的签名定义如下：对于每一对相邻的数字来说，如果第二个元素比第一个大，那就写下 I，否则写下 D，形成的字符串就是排列的签名。给出一个签名（长度不大于 1000），计算有多少个排列与其匹配。注意签名中可能包含“?”字符匹配 I 或者 D 皆可。输出排列个数除 1000000007 的余数。

火星老板（The Boss on Mars, Asia - Dalian 2011, LA5701）

公司有编号 1～n 的 n（$1\leqslant n\leqslant 10^8$）个员工，编号 k 的员工年薪为 k^4。老板为了节约开支，要辞掉编号与 n 互素的所有员工。计算这些被辞掉员工的年薪之和，因为数字较大，输出其模 1000000007 的余数。

提示：

n=4：1+3×3×3×3=82
n=5：1+2×2×2×2+3×3×3×3+4×4×4×4=354

棋盘（Chess Board, Asia - Dalian 2011, LA5702）

有一种 n 行×m 列的棋盘，格子是白色的，且大小一致。相邻格子之间是宽度一致的黑色边框，棋盘的边缘也是同等宽度的黑色边框，边框的宽度 b 不超过格子的宽度 a，如图 4.41 所示。

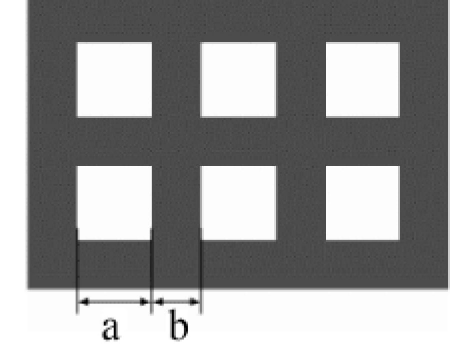


图 4.41

给出一个分辨率为 $H\times W$（$3\leqslant H,W\leqslant 2000$）的黑白图像，以及每个像素的颜色。要把这个图像涂成 n 行×m 列（$1\leqslant n,m\leqslant 200$）上述棋盘样式，每次只能选择一个矩形区域把其中所有像素涂成同一颜色，而且图像中同一个像素最多只能涂一次，涂一个矩形耗时 T。计算要把图像涂成指定大小棋盘的最小耗时。如果无法涂成棋盘，输出-1。

ACM/ICPC Asia – Tianjin（天津赛区）

加油站（Charge-station, Asia - Tianjin, 2012, LA6386）

有 n（$0<n\leqslant128$）个城市，女王要开车从城市 1 出发，一次性访问 n 个城市再返回 1。车在加满油之后只能走 D 米，每个城市都可以建一个加油站，在第 i 个城市建的费用为 2^{i-1}。城市可以重复经过。要保证女王有一条路线能够完成旅行，输出建加油站的最小费用。

输出的答案是二进制：如果在第 i 个城市建立加油站，答案从右往左数第 i 个值就为 1。第 i 个城市的坐标为 x_i,y_i（$0\leqslant x,y\leqslant1000$），城市 i 和 j 之间的距离为 $\mathrm{ceil}(\mathrm{sqrt}((x_i-x_j)^2+(y_i-y_j)^2))$，ceil 表示向上取整。

猎人（Hunters, Asia - Tianjin, 2012, LA6389）

Alice 和 Bob 都是最好的猎人，他们要比谁更强。现有虎狼各一，各住森林南北边。杀虎者得 X 分，杀狼者得 Y 分。杀掉两者得 $X+Y$（$1\leqslant X, Y\leqslant1000000000$）分。比赛开始前，Alice 在森林东边，Bob 在西边。比赛一开始，各选一个目标，但不知对方的选择。可能出现两种情况：

（1）二人选择不同目标，一定都能杀掉目标。

（2）二人选择同一目标，目标被 Alice 杀掉的概率是 P，被 Bob 杀掉的概率是 $1-P$。那么他们就去杀下一目标，这个目标被二人杀掉的概率也是 P 和 $1-P$。

但是 Alice 了解 Bob，她知道 Bob 选择虎作为第一个目标的概率是 Q，狼就是 $1-Q$。

输入 P 和 Q（$0\leqslant P, Q\leqslant1$，小数点后最多两位），计算出 Alice 应该选择虎还是狼的期望得分更高，以及最高得分的期望值（保留小数点后 4 位）。输入保证选择虎狼的期望得分是不一样的。

无路可逃（No place to hide, Asia - Tianjin, 2012, LA6390）

现在要抓捕一个妄想统治全人类的疯狂科学家 Gneh。他现在的位置是(X_m,Y_m)。N（$1\leqslant N\leqslant1000$）个国际刑警在附近不同的位置包围。第 i 个刑警的初始位置是(X_i, Y_i)，最大的移动速度是 V_i m/s。Gneh 发明了一个火箭摩托车用来逃跑，但是这个摩托车只能沿直线逃跑，需要保证 Gneh 必须被抓到。可以假设，一旦 Gneh 开始逃跑，刑警就可以同时看到 Gneh 逃跑的方向立即启动开始抓捕。所有的坐标值以及速度值都是在[−1e5,1e5]内的浮点数。

需要计算这 N 个刑警能否保证一定抓捕成功，如果能保证一定成功，输出最少需要多少个刑警。需要注意的是，如果刑警和 Gneh 的位置无限靠近，就可以认为抓捕成功。

mmm2 将军（mmm2, Asia - Tianjin 2012, LA6391）

mmm2 喜欢旅行，这次来到了以肉包子著名的天津。天津有 N（$1\leqslant N\leqslant50000$）个街区。这 N 个街区分成几个区域。每个街区属于正好一个区域。区域内部，街区个数和连接街区的街道个数相等，且任意两个街区都联通。对于不同区域的街区来说，之间没有街道。

第 i 个街区内的包子店的美味程度用整数 W_i（$|W_i|\leqslant 10^6$）来表示，旅途中 mmm 希望进入每个路过的包子店。mmm 不想重复路过同一个街区，但只要愿意，她可以选择走地道。天津市的任意两个街区即使没有街道连接，也有地道连接。mmm 走地道的次数必须小于 K（$1\leqslant K\leqslant 10$）。

现在需要计算在上述限制条件下，旅程中，能够路过的包子店的美味程度之和的最大值。因为有地道，mmm 可以选择从任何一个街区开始旅程，也可以从任何一个街区结束旅程。

ACM/ICPC Asia – Changsha（长沙赛区）

赞和蜡烛（LIKE vs CANDLE, Asia - Changsha 2013, LA6619）

微博上的一个帖子，有 N（$1\leqslant N\leqslant 50000$）个账户转发，形成了一个有根的转发树。每个账户有自己的价值 V（$0\leqslant V\leqslant 1000$）和转发态度（赞或蜡烛），若态度是“赞”，价值加到“赞”的一边，反之亦然。Edward 可以从“赞”的一边拿出 X（$0\leqslant X\leqslant 1000$）的价值去翻转一个账户，即把它的态度换到相反的一边。但是 Edward 发现，有的账户已经被别人翻转过了，对于这些，就要花费 Y（$0\leqslant Y\leqslant 1000$）的价值去翻转。一旦一个账户被翻转，树上转发这个微博的所有子账户也会被翻转。

原始帖子没有态度，也不允许被翻转。计算“赞”的一边与“蜡烛”一边的价值总数的最大差值。若最大差值为负数，则输出“HAHAHAOMG”。

Alice 的打印服务（Alice's Print Service, Asia - Changsha 2013, LA6611）

打印店的收费是阶梯式的，一次打印超过 $s1$ 页但不超过 $s2$ 的每页收费 $p1$，超过 $s2$ 不超过 $s3$ 的每页收费 $p2$…数据保证 $0=s1<s2<\cdots<sn\leqslant 10^9$，$10^9\geqslant p1\geqslant p3\geqslant\cdots\geqslant pn$。计算打印 q 页的资料最少花多少钱？例如 $s1=0$，$s2=100$，$p1=20$，$p2=10$ 时，若要打印 99 页，显然直接打印 100 页要更便宜一点，结果是 1000。

每个输入案例包含 m（$0 < n, m\leqslant 10^5$）个不同的 q（$0\leqslant q\leqslant 10^9$）。（$0=s1 <s2 <\cdots<sn \leqslant 10^9$，$10^9\geqslant p1\geqslant p2\geqslant\cdots\geqslant pn\geqslant 0$）。求出对于每个 q，如果要打印 q 页，输出要付出的最少费用（以分为单位）。

ACM/ICPC Asia – Nanjing（南京赛区）

可怜的仓库管理员（Poor Warehouse Keeper, Asia - Nanjing 2013, LA6633）

有一个仓库安装了货物记录器：其屏幕有两行显示，每一行有一个按钮。第一行表示数目，第二行表示总价。

按下第一行按钮单价不变，数量加 1，同时第二行的总价会根据当前的单价相应增加。例如，显示为 2,5，按下之后变成 3, 7（总价由 $2.5\times 2=5$ 变成 $2.5\times 3 = 7.5$）。

按下第二行的按钮数量不变，总价加 1，这样实际的单价就增大了。对于 3,7 来说，按下第二行按钮就变成 3,8，此时总价为 8.5。

屏幕上初始显示两个 1。要求变成第一行是整数 x（$1 \leqslant x \leqslant 10$），第二行是整数 y（$1 \leqslant y \leqslant 10^9$）。计算出要把初始的数字变成 x,y 至少需要按多少次按钮。如果问题无解，则输出-1。需要注意的是，总价可能是小数，但是第二行显示的是总价的整数部分。

样例输入：

```
1 1
3 8
9 31
```

样例输出：

```
0
5
11
```

对于第二个输入案例，一种可能的顺序是：(1,1)→(1,2)→(2,4)→(2,5)→(3,7.5)→(3,8.5)。

Cirno 的礼物（Cirno's Present, Asia - Nanjing 2013, LA6639）

给出一颗包含 N（$1 \leqslant N \leqslant 300$）个结点的树，你可以给每个结点等概率地染成 A、B、C 这 3 种颜色之一。对于一条边，如果两个端点颜色不一样，则断开这条边。对于一个特定的颜色，记 X 为包含奇数个结点的联通块个数，Y 为包含偶数个结点的联通块个数，则需要把这个颜色的所有联通块连接到一起需要的能量为 max(0, X−Y)。这个能量的数学期望乘以 3^N 一定是一个整数。计算要把 3 种颜色的联通块各自连接在一起需要的总的数学期望 E，输出 $E \times 3^N$ 模 10^9+7 的余数。

颜料混合（Wall Painting, Asia - Nanjing 2013, LA6640）

画家有 N（$1 \leqslant N \leqslant 10^3$）包颜料，每种颜料都有色值（用整数表示，不超过 10^9）。然后在 1～n 天中的第 k 天，他都会随机选择 k 包颜料进行混合，形成新的颜料，其色值是所有 k 个色值的按位取异或（xor）的结果。

输入 N 以及 N 种颜色的色值，计算每一天所有可能合成的颜料色值的总和，输出其模 10^6+3 的余数。

例如，N=3, k = 2，3 包颜料的颜色是 2、1、2。使用 3 种不同的两种颜色的组合，可以得到 3、3、0 这 3 种颜色。所以第二天所求的总和就是 3+3+0 = 6。

样例输入：

```
4
1 2 10 1
```

样例输出：

```
14 36 30 8
```

在样例中，第 1 天只能选择 1 种颜色进行混合，可能合成的值就是 1+2+10+1 = 14。

第 2 天可以选择 2 种：(1^2) + (1^10) + (1^1) + (2^10) + (2^1) + (10^1) = 36。

第 3 天可以选择 3 种：(1^2^10) + (1^2^1) + (1^10^1) + (2^10^1) = 30。

第 4 天可以选择 4 种：(1^2^10^1) = 8。

三色球摆放（Ball, Asia - Nanjing 2013, LA6641）

有红黄蓝 3 种颜色的球分别是 R、Y、B 个（每种球的个数不大于 10^9）。要把它们放成一排，每次放一个，位置随机。每次放球的得分规则如下：

- 第一个球得 0 分。
- 放两端，则得分等于放球之前一排球中不同的颜色个数。
- 放在两球之间，得分等于放球位置的两边两个队列中不同的颜色个数之和。

给出 3 种球的数量，计算出可能的最大得分。

ACM/ICPC Asia – Guangzhou（广州赛区）

火车排班（Train Scheduling, ACM/ICPC Asia – Guangzhou 2014, LA7074）

在一条单轨铁路线上有编号为 0～N 的 N+1 个站点（1≤N≤10），运行编号为 0～M−1 的 M（1≤M≤10）辆火车。每辆火车都有始发站 O、终点站 T、预计出发时间 E 以及最高限速 L（km/分钟）。一开始它停在始发站，在不早于预定时间的时间点出发开往终点站。途中每一站都要停，不同的列车限速不同，而且都不能超速。相对于整条铁路的长度，站点和列车都可以认为是一个点。

相邻两站之间的铁路称作一段（不含站本身）。两个相邻的站之间距离刚好都是 S（1≤S≤1000）km。每个站都能停下任意多个列车，但每一段都只有一个铁轨，只能跑同向的列车。跑在一段上的列车必须满足：

（1）方向相同。

（2）一列车可以追上但不能超过另一列。

列车的调度策略如下：

（1）忽略未出发和已经到达终点站的列车。

（2）当列车要从始发站出发，或者到达一个终点站之外的车站时，要立刻停下来等待进入下一段。

（3）当列车停在某站，只有以下两个条件满足它才能出发：① 这一段上没有对向驶来的列车。② 在段的两端没有编号更小的正在等待的列车。

（4）列车必须尽量顶着它的最高速跑，前提是不能同向超车。

按照编号的递增顺序依次输出按照以上的调度策略，每辆列车到达终点站的时间。结果要四舍五入。

样例输入：

```
2
1 3 100
0 1 0 5
0 1 20 5
1 0 0 5
2 2 100
0 2 0 4
0 2 2 5
```

提示：

在第 1 个案例中：
第 0 分钟，火车 0 和 2 都要出发开进同一段。根据我们的策略，0 先出发，2 要在站 1 等着，同时 0 在跑。在第 20 分钟，0 到达终点站。此时火车 1 和 2 都要开进同一段。1 先出发，2 要继续等待。
在第 2 个案例中：
火车 0 在第 0 分钟出发直接开往终点站。火车 1 的情况更复杂，它在第 2 分钟出发并且在第 10 分钟追上火车 0，因为无法超车，它需要一直跟着 0。他们会在第 25 分钟到达站点 1。接着编号更小的火车 0 先出发，火车 1 必须再次跟着 0 行驶。

ACM/ICPC Asia – Shanghai（上海赛区）

压制之刃（Quelling Blade, Asia – Shanghai 2011, LA5712）

有一种电脑游戏中，每种武器都要花钱购买，之后给角色带来一些加成。如果买了两件武器（不管是不是同类型），加成值就是累加的。例如，买了两件加成分别为 3 和 5 的武器，得到的总加成是 8。

每一种武器有一些依赖的其他武器。要购买这件武器，必须事先拥有所有的依赖武器。例如，购买武器 D，需要一个 E 和 S。如果再买 D，就需要再买一个 E 和一个 S。注意，已有的武器在购买之后不会消失。而且一个武器可能依赖多个同种类的武器，并且可以假设一种武器会被最多一种武器依赖。

每 1 秒钟，游戏会给角色 1 元钱，如果想要购买 Q，达到这个目标所需要的时间就很容易算出来。你的角色不仅想尽快拿到武器，也想最大化利用得到武器之前的每一秒。定义角色的功用为每一秒的加成值的和，计算到拿到 Q 武器的前一秒截止。也就是说，需要设计一个购买武器的流程使得拿到武器的时间最小化，并且最大化功用值。

给出游戏中的 N（$1 \leqslant N \leqslant 1000$）种编号为 1～$N$ 的 N 种武器，其中 Q 的编号是 1。每一种武器给出其加成值 B 和价格 C（$1 \leqslant B,C \leqslant 2^{31}-1$）。然后给出每种武器的 P 个依赖，每一个依赖给出编号 I 和数量 A。可以假设 Q 依赖的武器总数量小于 1000000，并且可以在有限的游戏时间内买到 Q。



计算出买到 Q 时间最小化的前提下，功用的最大值。可以假设结果能用 64 位有符号整数存放。

ACM/ICPC Asia – Chengdu（成都赛区）

传感器放置（Detector placement, Asia - ChengDu 2010, LA5007）

一个发射激光的点光源，坐标$(x0,y0)$，光线要经过$(x1, y1)$。放置一个和坐标平面垂直的三棱镜，给出 3 个顶点的坐标以及折射率 u（$1<u\leqslant 10$），空气的折射率是 1。所有坐标值不超过 1000，所有的点都在 x 轴上方，点光源不会在三棱镜内。计算激光最终和 x 轴交点的 x 坐标（精确到小数点后 3 位）。

光线折射相关的计算公式请参考图 4.42。

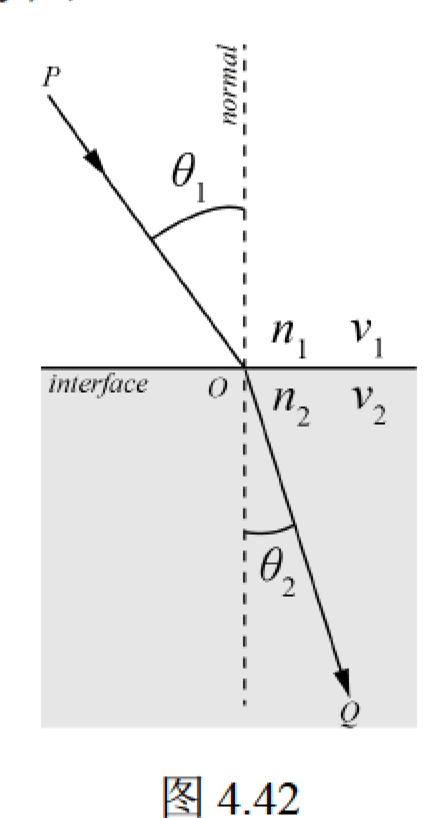


图 4.42

$\dfrac{\sin\theta_1}{\sin\theta_2}=\dfrac{v_1}{v_2}=\dfrac{n_2}{n_1}$，图 4.42 中 n_1 和 n_2 指的是两种介质的折射率，而 v_1 和 v_2 指的是两种介质中的光速。

双迷宫（Double Maze, Asia - ChengDu 2010, LA5008）

在双迷宫游戏中，需要用一系列的指令来同时走出两个迷宫，如图 4.43 所示。

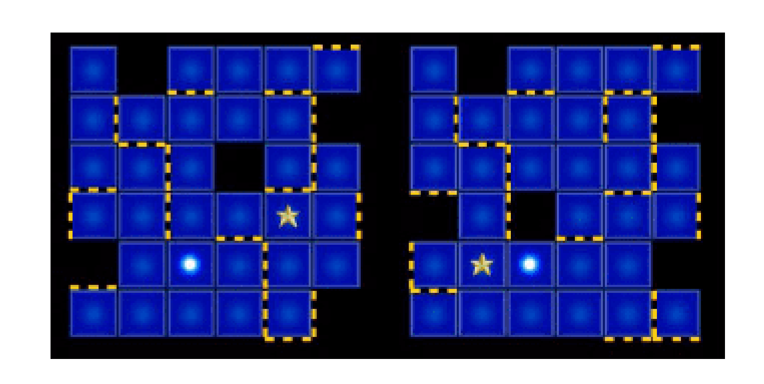


图 4.43

迷宫是一个 6*6 的方阵。格子是空洞或者方块。格子的 4 个边都可能有障碍，只有一个用圆球标记的出发点以及用五角星标记的目的点，二者可能重合。要圆球从出发点移到目的点。每一步共有 4 种命令，上下左右分别对应“U”“D”“L”“R”。如果命令方向有障碍，执行完之后无任何效果。如果球掉进空洞或者走出迷宫外，游戏失败。

双迷宫游戏中，命令同时控制两边的球，需要两个球都走到格子的目的点才算赢。计算能赢得游戏的最短命令序列，如果有多个则输出字典序最小的。如果问题无解，输出“-1”。注意，输入格式较为特殊，详情请参考原题。

积木构建游戏（Jenga, Asia - ChengDu 2010, LA5011）

Jenga 是一个考验动手能力和思维能力的游戏，游戏中玩家交替从积木塔中抽出一块积木并且使其平衡地放到塔顶，去创造一个不断增高且越来越不稳定的积木塔，直到积木塔倾倒。

游戏的材料是 54 个木块，块的长宽比都是 3:1。一开始，被叠成 18 层，每一层都是 3 个木块的长边在一起拼成一个正方形，并且不同层之间的块的方向互相垂直。如第一层是南北方向，第二层就是东西方向。

塔建好之后，玩家轮流出手。每一步都从任一层取出一块并且把它放到顶层上。但是顶层上的块不能被取出，如果顶层还没放完，次顶层也不能取。顶层要尽量放成和其他层一样，塔不倒这一步就成功了。

塔倒了游戏就结束，或者无论取哪块都会把塔弄倒（极少发生），游戏也结束。轮到谁先出手时游戏结束，谁就输掉。

对于任一层的木块来说，只有 4 种可能的布局，如图 4.44 所示（也可能是旋转了 90°的）。一开始都是 A，拿掉一块之后就会变成 B 或者 C 或者 C 的镜像。B 里面移走任何一块塔都会倒。对于 C 来说可以拿走一块变成 D。然后就不能再拿了。所以只有 3 种移动：（1）A→B；（2）A→C；（3）C→D。拿掉的木块就加到顶层了。

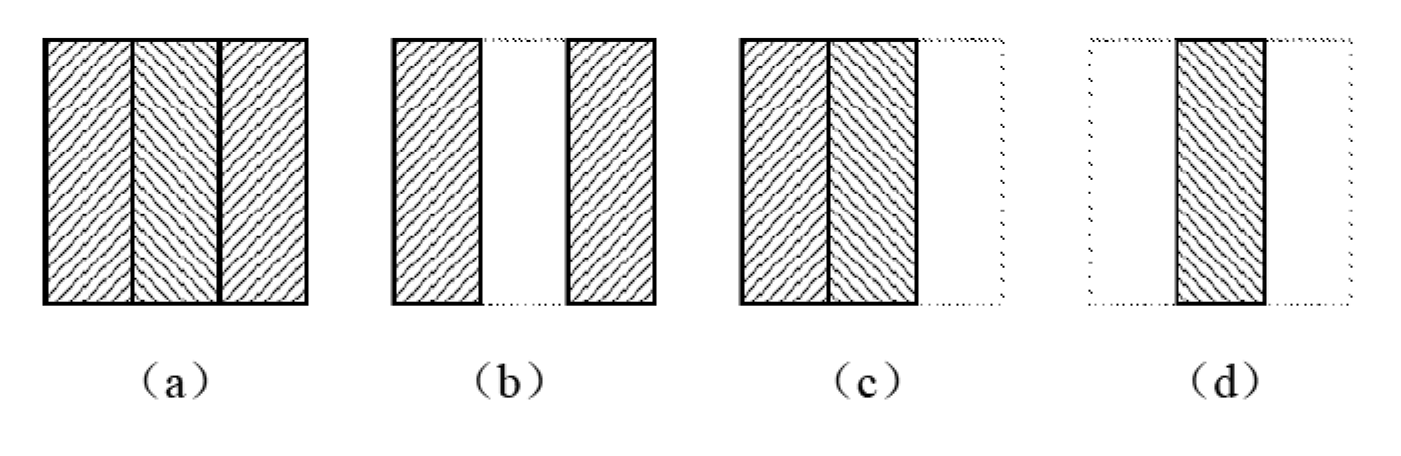


图 4.44

Alice 和 Charles 都是非常熟练的玩家，他们每一步的成功率都非常稳定，可以用公式 $P = b - d*n$ 来表示。其中 b 是某类移动的基准成功率，d 是每多一层成功率的减值。n 是移动之前的层数（包括顶部未完成那一层）。例如，一开始有 18 层，玩家的能力都是 $b = 2.8$，$d = 0.1$，那么第一轮 $P = 1.0$，第 2 轮和第 4 轮就变成 P=0.9。如果 P 不在区间[0,1]内，那么就使用区间内最接近的数字。例如，前几轮玩家不会失手，直到 n 稍微大些之前，P 会超过 1。

给出游戏开始时的层数 n_0（$3 \leqslant n_0 \leqslant 18$）。注意即使 $n_0 \neq 18$，游戏规则不变。然后给出 6 个实数：$b_{a1}, d_{a1}, b_{a2}, d_{a2}, b_{a3}, d_{a3}$，表示 Alice 在进行 3 种移动（A→B；A→C；C→D）的基准成功率和每一层的减值。同时也给出 Charles 对应的成功率和减值。每一对 b,d 都满足（$0 \leqslant b - d*n_0 \leqslant 2$ 且 $0<d \leqslant 0.5$），小数点后都不超过 4 位。

假如 Alice 先手，计算其取胜的概率，输出结果保留小数点后 4 位。

拯救公主（Rescue, Asia - ChengDu 2010, LA5012）

有标记为 $s_1,s_2,\cdots,s_n$ 的 n（$1\leqslant n\leqslant 50000$）个石头，从左到右排成一线。其中 S_i 的魔法值为 m_i（$1\leqslant m_i\leqslant 10^9$）。你有一种技能：站在石头 s_i 的右边向左发射 1 个初始能量为 p 的球，就会对每个石头 s_j（$j\leqslant i$）造成 $\mathrm{Max}(0,\ p-(i-j)^2)$的伤害，同时球的能量也要减少这个数值。从这个公式可以看出，对石头 j 的伤害仅仅取决于球的初始能量以及 j 和 i 间的石头个数。

如果收到的伤害总值大于魔法值，这个石头就算被毁掉。注意即使毁掉，石头不会消失，你的能量球依然会对它造成伤害并且能量值减少，也就是说，后面的球受到的伤害依然不变。你最多能发射 k（$1\leqslant k\leqslant 100000$）个能量球，但这个前提是要把 N 个石头全部毁掉，计算 p 的最小值。每次可以站在任意的位置发射，且 p 必须是正数。

相似度（Similarity, Asia - ChengDu 2010, LA5013）

对于一个单词序列{Tiger, Panda, Potato, Dog, Tomato, Pea, Apple, Pear, Orange, Mango}，可以分成 3 类，并且给每个单词按照分组用标签来表示：{A,A,B,A,B,B,C,C,C,C}。表示 Tiger、Panda、Dog 属于 A 组，Potato、Tomato、Pea 属于 B 组，Apple、Pear、Orange、Mango 属于 C 组。

但是标签本身并无意义，只是用来表示不同的分组方案，所以{P,P,O,P,O,O,Q,Q,Q,Q}和 {E,E,F,E,F,F,W,W,W,W}与上述表示实际上是等价的。但{A,A,A,A,B,B,C,C,C,C}和{D,D,D,D,D,D,G,G,G,G}就不等价。

定义两种表示 S 和 T 的相似度如下：$Similarity(S, T) = sum(S_i == T_i) / L$。其中 L 是 S 和 T 的长度，$sum(S_i == T_i)$表示所有标签等价的位置的个数。

给出长度为 n（$0<n<10000$）的单词序列的正确表示 S 和 m（$0<m<30$）个其他的表示 T，序列中的标签是刚好 k（$0<k<28$）个不同的大写字母。求出 S 和所有 T 的等价表示的相似度的最大值。

帝国时代（Age of Empires, Asia - Chengdu 2012, LA6365）

帝国时代游戏中有 4 种资源：食物、木头、石头和黄金。要使用农民来采集这些资源。一开始有 N 个农民，并且没有任何资源。每一秒钟，农民可以收集 A_1 个食物或者 B_1 个木头或 C_1 个石头或者 D_1 个黄金。注意同一秒钟内，农民只能采集一种资源。另外，只有在一秒钟结束时，资源才能采集到。但是不同的农民可以同时采集不同的资源。

也可以在游戏的过程中训练新的农民。需要在每一秒钟开始时投入 X 单位的食物，T 秒之后一个新的农民就开始工作了。注意开始训练时，你的食物必须大于 X 单位，而且同一时刻只能训练一个农民。

现在需要计算，要采集到 A_2 个食物、B_2 个木头、C_2 个石头以及 D_2 个黄金，最少需要多少秒。数据范围：（$1\leqslant N, X, T\leqslant 10^5$），（$1\leqslant A_1, B_1, C_1, D_1\leqslant 10^{18}$），（$0\leqslant A_2, B_2, C_2, D_2\leqslant 10^{18}$）。

斐波那契树（Fibonacci Tree, Asia - Chengdu 2013, LA6540）

给出一个包含 N（$1\leqslant N\leqslant 10^5$）个结点和 M（$1\leqslant M\leqslant 10^5$）条边的无向图 G。每一条边都是白色或者黑色的。计算能否找到一个恰好 F 条白色边的生成树，其中 F 必须包含在斐波那契数列（1, 2, 3, 5, 8, …）中。

随机数字游戏（Just Random, Asia - Chengdu 2013, LA6544）

庞老师和杨叔叔喜欢数字，每天早上他们玩一个游戏：

（1）庞老师在$[a,b]$内随机选择一个整数 x。

（2）杨叔叔在$[c,d]$内随机选择一个整数 y。

（3）如果 $x+y\equiv m(\bmod p)$，那么当天他们一起玩，否则各自回家。

给出整数 a, b, c, d, p 和 m（$0\leqslant a\leqslant b\leqslant 10^9$，$0\leqslant c\leqslant d\leqslant 10^9$，$0\leqslant m< p\leqslant 10^9$）。庞老师希望知道他们能够一起玩的概率，以最简分数来表示。

ACM/ICPC Asia – Hangzhou（杭州赛区）

子串（Substrings, Asia - Hangzhou 2012, LA6373）

给出一个长度为 n 的数组。需要计算对于给定的 w，所有长度为 w 的子串中的不同元素的个数之和。例如，数组是{1 1 2 3 4 4 5}。当 w=3 时，有 5 个长度为 3 的子串，它们分别是(1, 1, 2),(1, 2, 3),(2, 3, 4),(3, 4, 4),(4, 4, 5)。各自的不同元素的数目是 2,3,3,2,2。所以总和就是 2+3+3+2+2 = 12。

数据范围：$0<w\leqslant n\leqslant 10^6$，每一个数组会有 Q（$0\leqslant Q\leqslant 10^4$）个 w 需要计算，数组中的元素：$0\leqslant a_1, a_2\cdots a_n\leqslant 10^6$。

家庭作业（Homework, Asia - Hangzhou 2012, LA6377）

×××经常拖他的作业。这次他希望计算作业能否按期完成，作业有 n（$2\leqslant n\leqslant 5$）项，每一项都有一个期限 d（$1\leqslant d\leqslant 1000$），表示在第 d 个小时结束之前必须做完。对于每项作业，有一个耗时区间$[s_1,s_2]$（$1\leqslant s_1\leqslant s_2\leqslant 200$），表示如果要完成这项作业，花费最少 s_1 最多 s_2 小时的时间。用随机变量 t（可能是实数）来表示这项作业需要 t 小时完成。

（1）$s_1<s_2$，t 就是在区间$[s_1, s_2]$内等概率随机取。概率密度函数就是

$$f(t)=\begin{cases}\dfrac{1}{s_2-s_1}, & t\in[s_1,s_2]\\ 0, t\notin[s_1,s_2]\end{cases}$$

。

（2）$s_1=s_2$，则 $t=s_1$。

×××不能同时做多个作业，他希望合适地安排所有作业的顺序，使得所有作业能够按时或者提前完成。现在需要计算，所有作业能够按时完成的概率（用最简分数来表示）。如果概率为 0，输出“Poor boy!”。如果概率为 1，则输出“Congratulations!”。

ACM/ICPC Asia – Jinhua（金华赛区）

迷路（Lost, Asia - Jinhua 2012, LA6329）

彪哥来游乐园玩，有 N（$5\leqslant N\leqslant 100000$）个娱乐设施，还有 N 条双向道路连接这些设



施。你可以从其中任意一个开始然后再到其他的。每到一个设施，他把它标记为已访问。然后在连接到这个设施的未访问设施中选择一个，如果有多个，那么就等概率随机选择。

一开始要在 N 个娱乐设施中随机等概率选择，并且重复这个过程，直到无路可走。计算对于每个设施，彪哥最后一个访问的概率，然后输出 5 个最大的概率之和，保留小数点后 5 位。

输入的设施和道路形成的图中，显然只会有一个环，输入这个环的长度保证在 3～30。

疯狂坦克（Crazy Tank, Asia - Jinhua 2012, LA6331）

在疯狂坦克游戏中，在水平坐标 0 处，高度为 H（$1 \leqslant H \leqslant 100000$）的台子上有一个坦克 L。游戏开始前，坦克炮只能选择一个发射角度（可以是任意的），游戏开始就不允许调整了。之后需要发射 N（$0 \leqslant N \leqslant 200$）发炮弹，第 i 发炮弹的初速是 V_i。在坦克的右侧，$(L1,R1)$（$0 < L1 < R1 < 100000$）区间内有一个敌军坦克，$(L2,R2)$（$0 < L2 < R2 < 100000$）区间内有一个友军坦克。如果炮弹打到闭区间$[L1,R1]$内，就认为击中了敌军坦克，如图 4.45 所示。

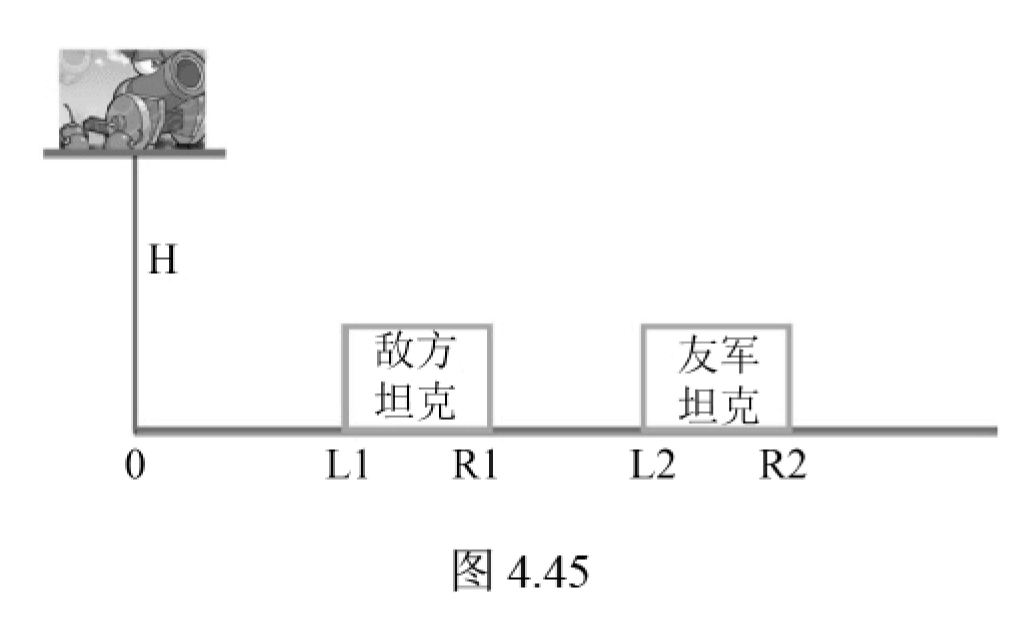


图 4.45

游戏的目的是选择一个发射角度，在保证不击中友军坦克的前提下增加击中敌军坦克的炮弹个数。计算这个最大的炮弹个数。本题中重力加速度 g=9.8，而且友军和敌坦克的区间可能重叠。

样例输入：

```
2
10 10 15 30 35
10.0
20.0
2
10 35 40 2 30
10.0
20.0
0
```

样例输出：

```
1
0
```

提示：

在第 1 个输入案例中，最优的选择是沿着水平线平行发射，那么第一个炮弹在 14.3 着陆，第二个在 28.6 着陆。
在第 2 个案例中，任何角度都无法保证在没有炮弹落在区间[2,30]内的前提下，有炮弹落在区间[35,40]内。

ACM/ICPC Asia – Taichung（台中赛区）

礼物问题（Present Problem, Asia - Taichung 2014, LA7000）

有编号为 0～n−1 的 n（1≤n≤10000）个人通过一个游戏来确定编号为 0～n−1 的 n 个礼物如何分配。游戏中有 n 条长度均为 L（1≤L≤10000）的垂直线，第 i 个人位于直线 i 的顶端，第 i 个礼物位于直线 i 的底端。还有 m（1≤m≤100000）条长度为 1 的横线连接相邻的两条直线，如(i,k)就表示连接 i 和 i+1 的距离顶端距离为 k（0<k<1）的横线。不会有两条相邻的横线(i,k)和(i+1,k)在同一个位置连接一条垂直线。

人从顶端往下走，遇到横线就必须横着走，碰到竖线之后继续往下走，遇到横线再横着走，走到最下面拿到礼物结束，如图 4.46 所示。

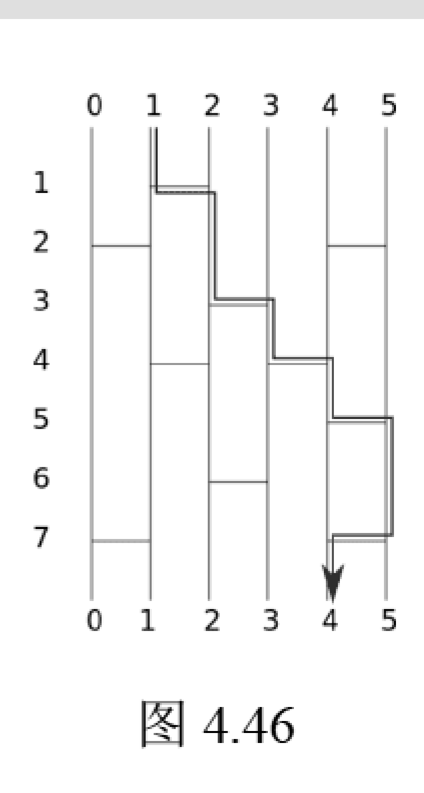


图 4.46

输出每个人拿到的礼物的编号。

树上的平衡游戏（A Balance Game on Trees, Asia - Taichung 2014, LA7003）

在一棵树上可以玩 K−平衡游戏。对于顶点 v 来说，如果它是白色的，并且恰好有 k 个相邻顶点是黑色的，那么 v 就是平衡顶点。如果有一个结点不平衡，那么就必须把它涂成黑色的。

例如，在图 4.47（a）中，k=1，那么平衡游戏的答案包含 3 个平衡顶点。在图 4.47（b）中，k=2，只能得到 1 个平衡顶点。

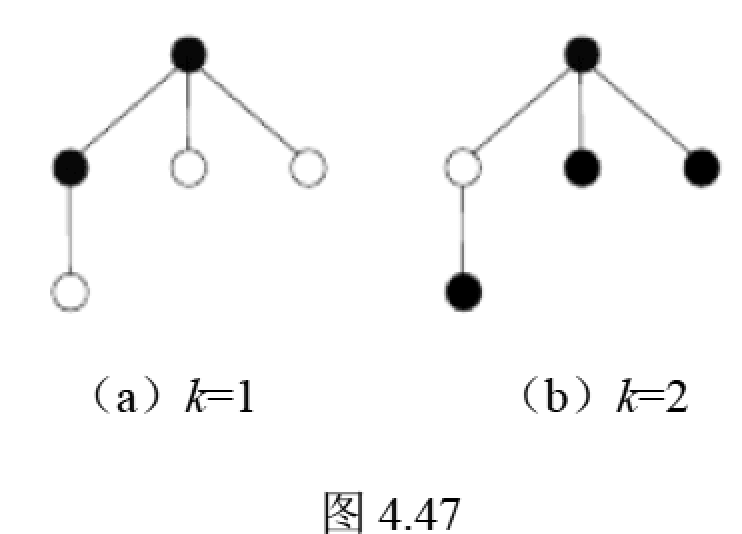


图 4.47

给出一棵树（顶点数目不超过 100）的结构以及 k（k≤10），顶点都是白色的。计算通过把某些顶点涂黑最多可把多少个顶点变成 k−平衡顶点。

ACM/ICPC Asia – Kaohsiung（高雄赛区）

增强现实游戏（AR Game, Asia - Kaohsiung 2010, LA5019）

有编号为（1～5）的 5 种标记，依次如图 4.48 所示，都是 100*100 像素的黑白图像。输入一个图像，判断和哪个标记匹配。这个输入图像规则如下：

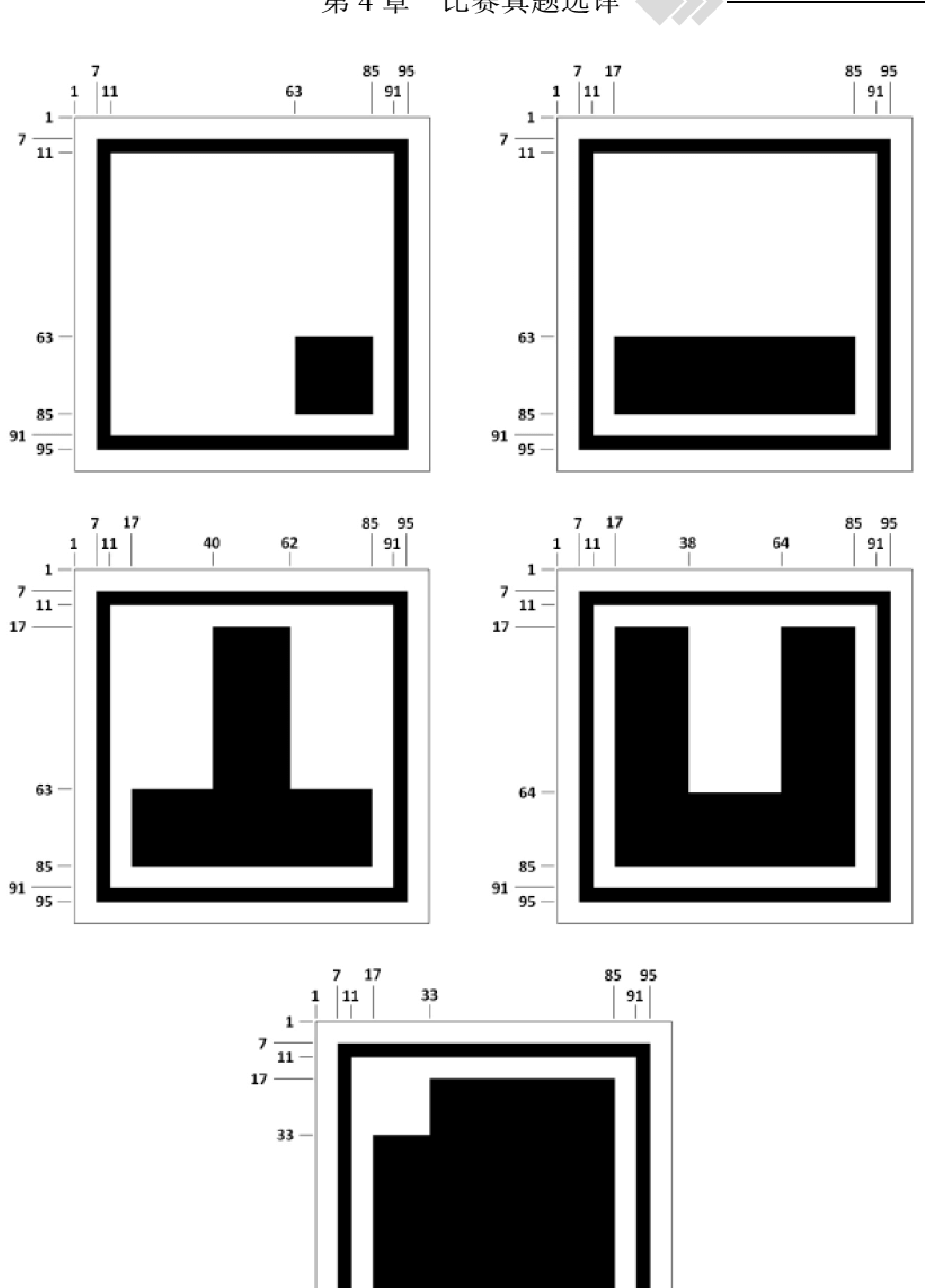


图 4.48

（1）都是二进制描述，0 表示黑像素，1 表示白像素。

（2）可能经过了 0°、90°、180°、270°旋转。

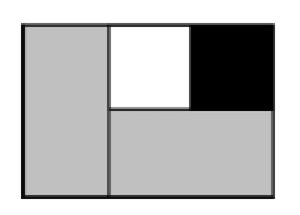
（3）可能放大缩小了，大小可能是 50×50、100×100、150×150 或 200×200。

（4）包含一些噪点，最多 3%的像素被黑白反转了。

输出可能和此图像匹配的标记名称。

建房（Houses, Asia - Kaohsiung 2010, LA5020）

有一个 n 行×m 列的方阵区域（$1\leqslant m,n\leqslant 200$），要在上面建房。每个房子占用横着或


图 4.49

竖着的两个连续单位方格，并且不能互相重叠。另外，区域内有 k 个坏块不能建房，给出 k 个坏块的位置，计算区域中最多能建几个房子。

图 4.49 就是一个 3×2 的区域，包含坐标为（2,1）的坏块不能建房，最多可以建两个房子。

树的表示（Tree Representation, Asia - Kaohsiung 2010, LA5021）

给出一个包含 n（$5 \leqslant n \leqslant 50$）个有标记（都是 1～$n$ 之间的唯一整数）结点的树 T，重复以下步骤直到只剩一条边：删除标记最小的叶子结点并记录下其父结点的编号。生成的 $n-2$ 个标记的序列就可以用来表示一棵树。一个结点的标记在最终的序列中出现的次数等于它的度数减一。

给出长度为 $n-2$ 的序列，从这个序列还原出整棵树。输入保证结果唯一。

接礼物游戏（Falling Gift Game, Asia - Kaohsiung 2010, LA5024）

游戏机示意图如图 4.50 所示。

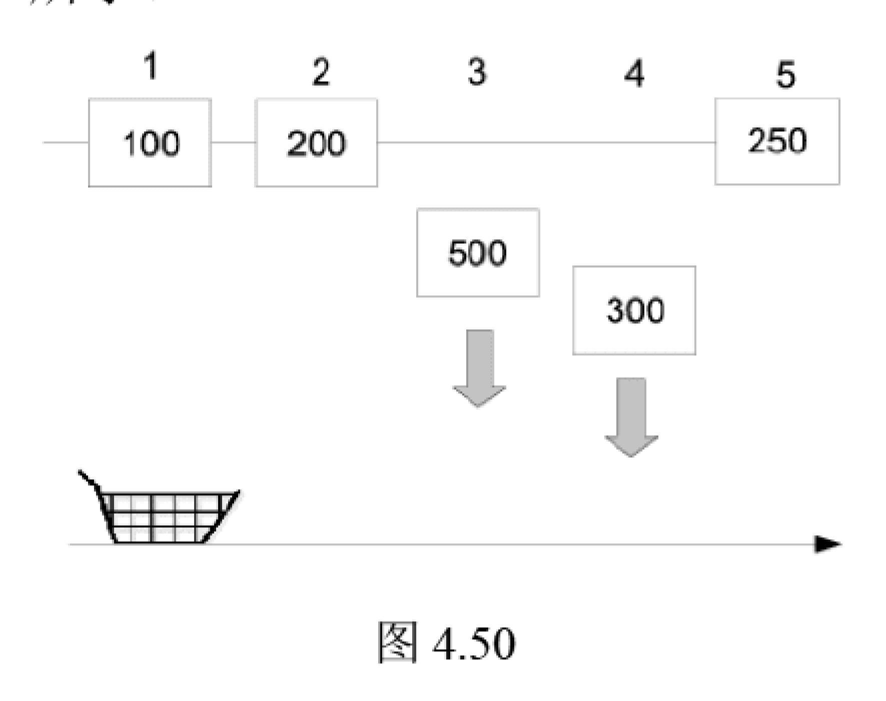


图 4.50

游戏机在指定的时间从每个位置落下一个礼物，你推个小车在底下的水平线上一开始在最左端位置 1 处，可以控制向右移动但是不能向左移动。相邻的两个位置间移动需要用时 1 秒，也可以停在一个位置等礼物落下。

礼物共有 G（$G<500$）个，按照位置从左到右的顺序，输入其落到水平线上的时间 t（$1 \leqslant t \leqslant 2G$）及其价格 p（$1 \leqslant p \leqslant 10000$）。计算你能接到的礼物总价的最大值。

ACM/ICPC Asia – Amritapuri（印度 Amritapuri）

黑暗骑士（The Black Riders, Asia - Amritapuri 2012, LA6339）

在夏尔的田野上有 N 个霍比人。黑暗骑士也在寻找指环，当霍比人听到骑士靠近或者感觉到骑士带来的恐惧时，他们就立刻逃进附近的 M 个地洞中去。

每个地洞只能容纳一个霍比人，一旦霍比人进洞，他就可以用 C 个时间单位挖出可容纳另一人的空间。而且即使挖过之后，一个洞也最多能容纳 2 人。注意 1 人只能在进洞之后开挖。输入 N,M,K,C，接下来 N 行，每一行包含 M 个整数，表示对应的霍比特人到达每个地洞的时间 W。计算让至少 K 个霍比人藏好所需要的最短时间。



数据范围：

- $1 \leqslant N, M \leqslant 100$。
- $1 \leqslant K \leqslant \min(N, 2M)$。
- $0 < C < 10000000$。
- $0 < W < 10000000$。

样例输入：

```
2
3 3 2 10
9 11 13
2 10 14
12 15 12
4 3 3 8
1 10 100
1 10 100
100 100 6
12 10 10
```

样例输出：

```
10
9
```

样例解释：

第 1 个案例中，有 3 个人 3 个洞，并且需要保证 2 个的安全。可以让#1 去第 1 个洞，#2 去第 2 个洞，需要 10 个时间单位。

第 2 个案例中，可以让#1 在时间点 1 到洞 1，#2 时间点 9 到洞 1（这个时候#1 已经挖好洞），在时间点 6 ，#3 到达洞 3。

Aglarond 的闪亮洞穴（The Glittering Caves of Aglarond, Asia - Amritapuri 2012, LA6345）

墙上有一些闪亮的钻石组成的 $N*M$（$1 \leqslant N, M \leqslant 50$）的网格，每个钻石后面都有个灯。对于每一行的灯都有一个开关可以切换这一排所有灯的开关状态。给出初始所有灯的状态，每次操作可以选择一行的开关（每一行都可以按任意多次）。计算通过正好 K（$1 \leqslant K \leqslant 100$）次操作，可以将最多几个钻石灯点亮。

样例输入：

```
2
3 3 2 10
9 11 13
2 10 14
12 15 12
4 3 3 8
1 10 100
```

```
1 10 100
100 100 6
12 10 10
```

样例输出：

```
10
9
```

样例解释：

- 在第 1 个案例中，可以将行 1 切换 1 次，把 4 个灯全部打开。
- 在第 2 个案例中，行 1 或者行 2 都可以被切换 2 次，然后回到初始状态。

萨鲁曼涂色（Saruman of Many Colours, Asia - Amritapuri 2012, LA6347）

萨鲁曼有一支 N（$1 \leqslant N \leqslant 20000$）个半兽人的军队，依次绑在传送带的椅子上。传送带经过涂色间时，可以前后两个方向移动。按照座椅的顺序，半兽人编号是 0 到 $N-1$，萨鲁曼希望其中第 i 个涂上颜色 $c[i]$（c 是长度 N 的小写英文字母字符串）。

涂色间只能放下 K（$1 \leqslant K \leqslant N$）个椅子，当 K 个连续的半兽人送到房间中，涂色机把它们都涂上同一颜色，传送带不是环形的。在这个过程中半兽人可以被重新涂色。萨鲁曼希望计算出涂色机最少需要使用几次才能把每个半兽人涂成他想要的颜色。如果问题无解，则输出“-1”。

样例输入：

```
2
3 2
rgg
3 3
Rgg
```

样例输出：

```
2
-1
```

样例解释：

（1）在样例 1 中，0 和 1 号半兽人可以先涂成“r”，接着 1 和 2 号可以涂成“g”。

（2）在样例 2 中，因为 $N=K$，所有半兽人只能被涂成同一个颜色，问题无解。

兽人（Orcs, Asia - Amritapuri 2012, LA6349）

兽人仅仅对他的直接上级负责，这样一直到军队的头领。如果一个兽人的直接上级死了，这个兽人就脱离体系变成无赖。有编号 1～N 的 N（$1 \leqslant N \leqslant 100000$）个兽人，其中头领的编号是 1。要进行点名来确认他们的忠诚度：

（1）随机对编号排序。

（2）按这个顺序进行点名，看看是不是已经死了。



（3）如果点到编号的兽人已死，标记为“已删除”。

这个点名过程可做一个优化：在第 2 步，如果兽人的任何一个上级已经被标记为“已删除”，那么点名就不做了。给出整个兽人军队的等级体系，考虑第 1 步中的所有可能排序，能通过上述优化节省的点名次数的数学期望是多少？输出精确到 10^{-6}。

样例输入：

```
2
2
1 2
1
1
2
1 2
1
2
```

样例输出：

```
0.5
0
```

样例解释：

（1）在案例 1 中，兽人 1 已死。所有可能的两种顺序是[1,2]和[2,1]。使用优化，对于顺序[1,2]来说，点名次数优化到 1。所以不加优化的点名总次数是 4，优化后是 3。所以点名次数优化的数学期望是(4−3)/2 = 0.5。

（2）对于案例 2 来说，兽人 2 已死。因为他没有任何下级，所以优化无效。

道路装饰（Road Decoration, Asia - Amritapuri 2014, LA6981）

有 N（$1 \leqslant N \leqslant 20000$）个场馆，其中第 0 个是中心场馆。连接这些场馆的是一个包含 M（$0 \leqslant M \leqslant 40000$）条道路的双向路网，路的长度 w 都满足 $1 \leqslant w \leqslant 10^9$。现在需要选择一些道路进行装饰涂色，使得每个场馆到中心场馆都只有 1 条涂色路径。

对涂色路径有两个需求：

（1）所有场馆到场馆 0 的路径长度之和最小。

（2）所有路径长度之和最小。

计算能否有一种方案可以同时满足以上两个需求。注意，如果有一个场馆到场馆 0 不连通，问题无解。

样例输入：

```
3
3 3
0 1 1
0 2 2
1 2 2
```

```
3 1
0 1 1
4 5
2 1 9
3 2 5
0 3 9
0 1 2
3 1 9
```

样例输出：

```
YES
NO
NO
```

样例解释：

第 1 个案例中，两种方案都可以选择对道路{0<–>1, 0<–>2}涂色。

第 2 个案例中，点 2 和 0 是不连通的。

Hugphile 顺序（Hugphile Order, Asia - Amritapuri 2014, LA6986）

MCG 要举办 2015 年世界板球总决赛，主办方要在一个垂直杆子的 n（$1 \leqslant n \leqslant 10^{18}$）个不同位置安装 n 个不同分辨率的摄像头，安装的顺序要根据 Hugphile 顺序决定。根据 Hugphile 顺序，如果摄像头在杆上的位置是 $p1 \cdots pn$，其中 pi（$i = 2 \cdots n$）位置的摄像头比在 $pi/2$ 位置上的摄像头有更高的分辨率（$pi/2$ 表示整数除法）。

假设 n 个摄像头的分辨率都在闭区间$[1,n]$内。给出一个分辨率为 m（$1 \leqslant m \leqslant n$）的摄像头，找到根据 Hugphile 顺序，它在杆子上所有可能安装位置的个数。测试案例的个数 T 满足（$1 \leqslant T \leqslant 10^5$）。

加巴冲刺（Gabba Sprint, Asia - Amritapuri 2014, LA6988）

有一个体育馆，周围有间隔相同的编号为 1～N 的 N 根柱子。前 $M-1$ 根柱子通向体育馆的入口，编号为 $M, M+1, \cdots, N$（$1 \leqslant M < N \leqslant 500$）的柱子以环形围绕着场馆，编号为 M 的柱子刚好与 N 相邻。V 和 R 两个人都是从杆子 1 开始跑步，每次都是从柱子 i 跑到 $i+1$。除了 $i=N$ 时，下一个杆子是 M，如图 4.51 所示。

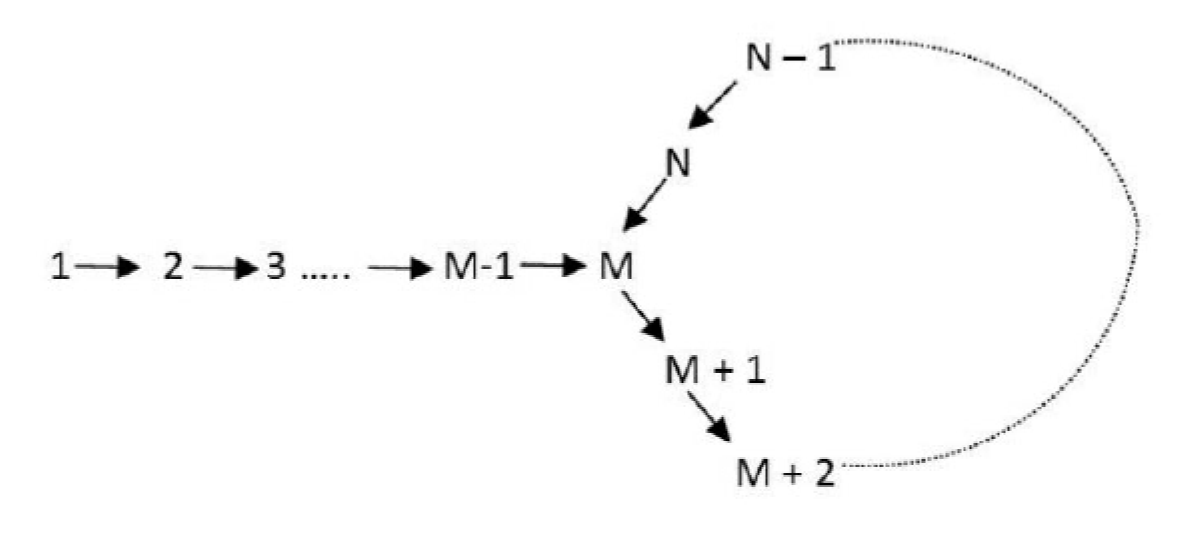


图 4.51

V 和 R 二人的速度分别是 P 和 Q（$1 \leqslant P, Q \leqslant 500$ 且 $P \neq Q$），意思是每分钟能跑过几根



柱子。他们要跑 S（$1 \leqslant S \leqslant 100000$）分钟，因为不能长时间奔跑，所以教练要求每次就短跑 1 分钟。一开始 V 跑一分钟，经过 P 根柱子停下来。然后 R 跑 1 分钟，经过 Q 根柱子停下。V 再次起跑，他们如此往复跑 2*S 分钟（每人跑 S 分钟）。V 和 R 如果在某个时间点停在同一根柱子，那么他们就算相遇。计算他们在整个比赛过程中能相遇几次？

输入样例：

```
2
4 1 4 2 3
5 3 10 1 2
```

输出样例：

```
1
3
```

样例分析：

在第 1 个样例中，V 在第 1、2、3、4 分钟之后位置分别是{3,1,3,1}，R 的位置是{4,3,2,1}，所以，他们就是在第 8 分钟（各自的第 4 分钟）在 1 号杆相遇。在第 2 个样例中，他们会在各自的第 3、6、9 分钟相遇。

ACM/ICPC Asia – Hatyai（泰国合艾）

终极设备（Ultimate Device, Asia - Hatyai 2012, LA6146）

组装一个设备，需要从 n（$1 \leqslant n \leqslant 100$）个电路中选择，其中第 i 个电路的燃烧周期是 t_i，会在第 $k*t_i$（k=1,2,3…，$1 \leqslant t_i \leqslant 500$）秒进入烧灼状态。在任一秒，只要有一个电路不在烧灼状态，那么设备就安全。如果所有电路都进入烧灼状态，那么设备就烧坏了。例如，设备有 3 个电路，烧灼周期分别是 3、4、5，那么在第 60 秒就会烧坏，设备的寿命就是 60。

在给设备挑选电路时，对于第 i 个电路，用抛硬币的方法确定是否选用，如果结果是正面，那就选用这个电路。n 次抛硬币之后，就选好了。

首先计算设备寿命的数学期望值 r。如果没有电路被选上，那么设备寿命就是 0。输出 (r*2n)mod 10007，如果 r*2n 不是整数，就输出“not integer”。

曲速前进（Warp Speed II, Asia - Hatyai 2012, LA6147）

曲速引擎有多种状态，靠在状态间的跳跃来推动飞船前进。0 是空闲状态。当且仅当在这个状态下，引擎不能跳跃。一开始引擎就在 0 状态。跳跃所需能量取决于引擎的当前状态。状态间切换也需要一定的能量。每次旅行，给出一系列的跳跃。需要找出一系列的引擎状态。引擎的终结状态也必须是 0。

输入包含空行分隔开的 4 部分。

第 1 部分给出状态的个数 N（$1 \leqslant N \leqslant 100$），状态 id 用 0～$N$−1 的整数编号。当且仅当状态 0 时，引擎不能跳跃，并且 0 也是任意输出状态序列的起始和结束状态。接着是跳跃

的种类数 H（1≤H≤1000）。跳跃 id 是 0～H−1。

第 2 部分使用一个 N 行×N 列的表格，给出在不同的引擎状态之前切换所需的能量（1≤能量值≤100）。

第 3 部分是一个 N 行×H 列的表格，给出每种状态下进行每种跳跃所需的能量（1≤能量值≤100）。注意表格的第 1 行，都是 0，表示状态 0 时不能进行任何跳跃。

第 4 部分包含一系列要执行的跳跃序列（1≤序列的个数≤1000）。包含 1 到 1000 行，每一行包含 1 个跳跃序列（1≤序列的长度≤1000），序列中包含空格分开的跳跃 id。

对于第 4 部分的每个跳跃序列，计算出完成这些跳跃所需的能量之和的最小值。接下来另起一行输出一系列的引擎状态（个数必须和输入跳跃序列的个数相同，每次跳跃之前都要输出其引擎状态）。如果有多重状态序列消耗的能量相同，输出字典序最小的那个状态序列。

样例输入：

```
4 5

1 2 6 1
3 4 3 17
2 3 9 3
1 21 1 8

0 0 0 0 0
3 3 2 4 3
2 2 4 3 1
4 2 2 7 7

0 4
1 2 3 2
```

样例输出：

```
9
3 2
23
1 1 2 3
```

样例解释：

样例输入中有 15 行，输出有 4 行。

在第 1 个样例输出（输入第 13 行的解），所需的最小能量是 9。引擎的状态序列是[3 2]或者说[0−3 2−0]。有 3 次状态切换：0→3,3→2,2→0，耗费的能量是 1+1+2=4。以及分别对应于状态 3 和 2 的两次跳跃（0 和 4），所需的能量是 4+1 = 5。所以总的能量是 4+5=9。

最后一个样例输出中，最小能量值是 23。问题的解是[0−1 1 2 3−0]，共需要能量是 23。



ACM/ICPC Asia – Bangkok（泰国曼谷）

音量控制（Volume Control, Asia - Bangkok 2014, LA6843）

电脑的音量控制有主控和从控，各自都有 N（$N \leqslant 30000$）个音量级别，则两个控制的级别叠加在一起就会有更多的级别。计算除了 0%之外，叠加在一起总共能设置出多少个音量级别。

例如，$N = 4$，则两个控制都能设置到 0%, 25%, 50%, 75% 以及 100%。则两者叠加在一起能设置出的级别是 0%, 0%, 0%, 0%, 0%, 0%, 6.25%, 12.5%, 18.75%, 25%, 0%, 12.5%, 25%, 37.5%, 50%, 0%, 18.75%, 37.5%, 56.25%, 75%, 0%, 25%, 50%, 75%, 100%。去重之后就是 0%, 6.25%, 12.5%, 18.75%, 25%, 37.5%, 50%, 56.25%, 75%, 100%，共 10 个。

扫雷者（Landmine Cleaner, Asia - Bangkok 2014, LA6849）

扫雷探测器要在一个 N（$1 \leqslant N \leqslant 1000$）行×$M$ 列（$1 \leqslant M \leqslant 1000$）的方阵区域内扫雷，方阵的每个格子中心都可能有 1 个地雷，也可能没有地雷。探测器在飞过每个网格时，读到的信号读数是这样的：

（1）如果这个网格有雷：3 +（8 个方向相邻网格的地雷个数之和）。

（2）无雷：0 + 8 个方向相邻网格的地雷个数之和。

给出机器人在每个格子收到的信号强度，计算每个格子中是否有雷。需要注意的是，网格方阵之外的区域无雷，输入保证答案唯一。

隐藏的加号（Hidden Plus Signs, Asia - Bangkok 2014, LA6850）

我们需要在一个 $R \times C$（$3 \leqslant R,C \leqslant 30$）的数字表中找出隐藏的加号，规则如下：

（1）加号可能互相覆盖。

（2）加号的长宽在 3～11 个单位之间，每个加号的长宽相等。

（3）每个单位相当于数字表的一个网格的宽度。

（4）每个表格中可能有 2～9 个加号，加号的区域绝对不会覆盖到表格外。

（5）加号的中心绝不会在其他加号的区域内。

（6）每个格子都有一个数值，表示覆盖这个格子的加号的个数。

举例来说，图 4.52 中有 2 个加号，长度分别为 3 和 5。中心坐标分别是(2,2)和(3,3)，高亮标出。

0	1	1	0	0
1	1	2	0	0
1	2	1	1	1
0	0	1	0	0
0	0	1	0	0

图 4.52

给出一个表格，计算出其中加号的个数，并且输出中心最靠右下方的加号的中心坐标。

样例输入：

```
2
5 5
0 1 1 0 0
1 1 2 0 0
1 2 1 1 1
0 0 1 0 0
0 0 1 0 0
10 11
0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 1 1 0 1 0 0 0
0 0 1 1 1 2 2 1 1 0 0
0 0 1 2 2 1 1 2 2 0 0
0 0 1 1 3 1 0 0 1 0 0
0 0 0 2 1 2 2 1 1 1 1
0 1 0 0 1 1 1 0 1 1 0
1 1 1 0 1 1 1 1 3 1 1
0 1 0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 1 0 0 0 0
```

样例输出：

```
2
3 3
9
8 10
```

提示：

样例输入 2 的棋盘中加号的中心点如图 4.53 所示。

0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0
0	0	1	1	1	2	2	1	1	0	0
0	0	1	2	2	1	1	2	2	0	0
0	0	1	1	3	1	0	0	1	0	0
0	0	0	2	1	2	2	1	1	1	1
0	1	0	0	1	1	1	0	1	1	0
1	1	1	0	1	1	1	1	3	1	1
0	1	0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0

图 4.53

毛毯（Blanket, Asia - Bangkok 2014, LA6852）

冬天要到了，需要给大家分配毛毯用来御寒。有编号为 $0\sim M-1$ 的 M（$1\leqslant M\leqslant 10^6$）个人排成一条直线。有 n（$1\leqslant n\leqslant 10^5$）条毛毯，每个都无限长，但只有特定部分的厚度足够

用来保暖。毛毯用两个整数(a,b)（$1\leqslant a\leqslant b\leqslant 16$）来描述，一开始是长度为 a 的厚的部分，接着是 $b-a$ 的薄的部分，然后无限地循环下去。例如，对于（2,3）的毛毯来说，第一个厚的部分能够覆盖在 0,1 位置的人，第二个厚的部分能够覆盖在 3, 4 位置的人，如此等等。

把所有毛毯摞起来放到人排成的直线上，从位置 0 开始摆。有些人就可能被多个厚的部分盖住，有的可能一个厚的部分都盖不到。计算分别被 0,1,2,⋯,n 个厚的部分盖住的人数。

城市（City, Asia - Bangkok 2014, LA6854）

有一个 N 行×M 列网格布局的城市，每个街区都是正方形的，之间都有大街，图 4.54 就是一个 4×6 的城市布局。

要从一个街区到相邻街区，必须走人行横道。人行横道连接水平或垂直相连的两街区。两街区之间可能有多个人行横道相连。如果一个人行横道 A 把街区 B 和其相邻的街区连接起来，那么称 A 属于 B。

给出属于除了某个街区之外的所有街区的所属人行横道个数。计算出这个街区的所属人行横道个数。

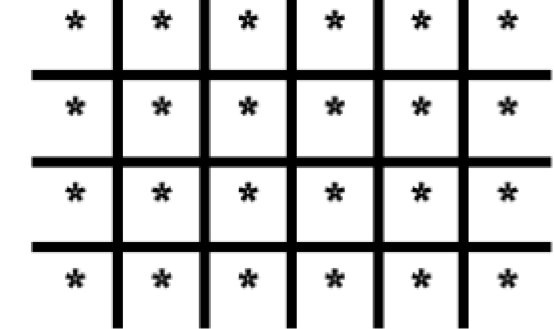


图 4.54

ACM/ICPC Asia – Phuket（普吉岛赛区）

快乐涂色（Fun Coloring, Asia - Phuket 2011, LA5725）

一个有限集 U 及其子集 $S_1, S_2, S_3, \cdots, S_m \subseteq$U 且$|S_i|\leqslant 3$。

问题：是否存在一个函数 f : U→{红, 蓝}使得对于每个 S_i，至少有一个成员的颜色跟其他成员不一样？给出一个 U=$\{x_1,x_2,x_3,\cdots,x_n\}$（$4\leqslant n\leqslant 22$），以及 $S_1, S_2, S_3, \cdots, S_m$（$6\leqslant m\leqslant 111$）。判断是否存在这样的函数 f。

大陆合并（Coalescing Continents, Asia - Phuket 2011, LA5729）

给出一些矩形块。判断这些矩形块能不能合并到一起形成一个完整的矩形。可以选择一个块，然后每次在“上下左右”中选择 1 个方向并移动一个单位。移动过程中，两个矩形块可以互相覆盖。但是在最终合并完成之后就不能互相覆盖了，而且最终的矩形中不能有洞。计算需要最少多少次移动。如果无法形成矩形，则直接输出“invalid data”。

输入数据是一个 20*20 的字符方阵。每个字符要么是点（.），表示是空的，要么是一个小写的“x”，表示是一个矩形块的一部分。

输入数据符合以下条件：

- 输入方阵中，不同的矩形块的 x 不会相邻。有公共边的格子才算相邻。
- 方阵中最多有 25 个 x。
- 矩形块的个数 K 不超过 5 个。
- 矩形块和目标矩形的边都是跟坐标轴平行的。

ACM/ICPC World Finals

渡轮（Ferries, World Finals 2002 - Honolulu, LA2477）

挪威海岸线有许多峡湾，所以旅行路线一般是开车和渡轮交替进行，开车时最高限速是 80km/h。现在给出旅行中每一段的信息：

（1）如果是开车，给出这一段距离（单位是 km）。

（2）如果是乘渡轮，则给出渡轮全程所需时间，每小时的班次数 f（f>0），每个班次出发时刻是几分。

```
Manhiller Fodnes ferry 20 2 15 35
```

举例来说，上述信息就表示从 Manhiller 到 Fodnes 需要乘渡轮 20 分钟到达，每小时 2 班，出发时刻是（0:15, 0:35, 1:15, 1:35…）。

计算需要最短多少时间到目的地（输出格式是 hh:mm:ss），并且在总时间最短的前提下要求开车时最高车速尽量低。输出最少的总时间和开车时的最高车速。

沿着总线（Riding the Bus, World Finals 2003 - Beverly Hills LA2723）

CPU 内的总线布局有一种称为 SZ 曲线，都是画在一个 1×1 大小的单元上，图 4.55 所示的 1 阶曲线，就是字母“S”的样子，由依次连接点(0,0), (1,0), (1,0.5), (0,0.5), (0,1)和(1,1)的线段组成。SZ 曲线中，水平线段的长度两倍于垂直线段。1 阶曲线中垂直线段的长度是 0.5。

图 4.56 的 2 阶曲线由 9 个更小的 1 阶曲线组成，其中 4 个做了水平反转，变成 Z 形。这些曲线由长度为 len 的虚线连接。2 阶曲线的边长为 8*len，并且整体宽度为 1，故 2 阶曲线中的 len = 0.125。从 k 阶曲线到 k+1 阶曲线的生成方法类似。对于 k 阶曲线来说，组件连接点是以 len 为间隔等距离分布在曲线上，共 9^k 个，曲线的长度是$(9^k-1)\times$len 个单位。

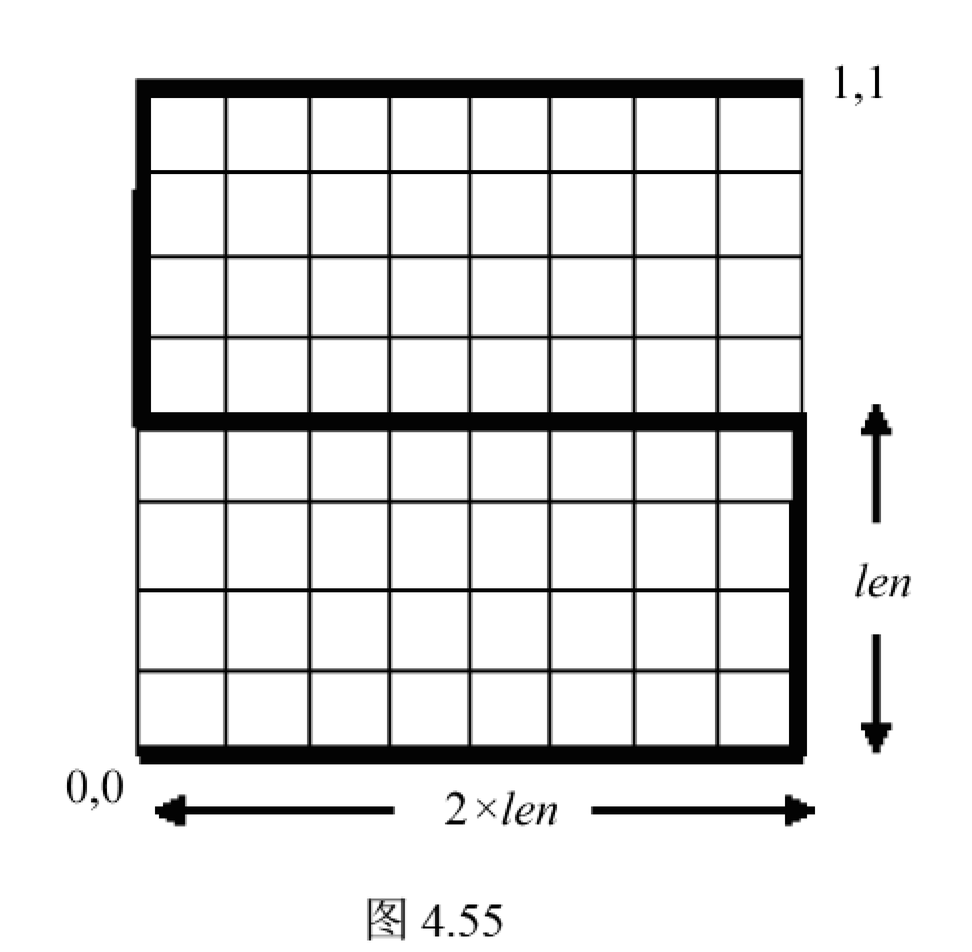


图 4.55

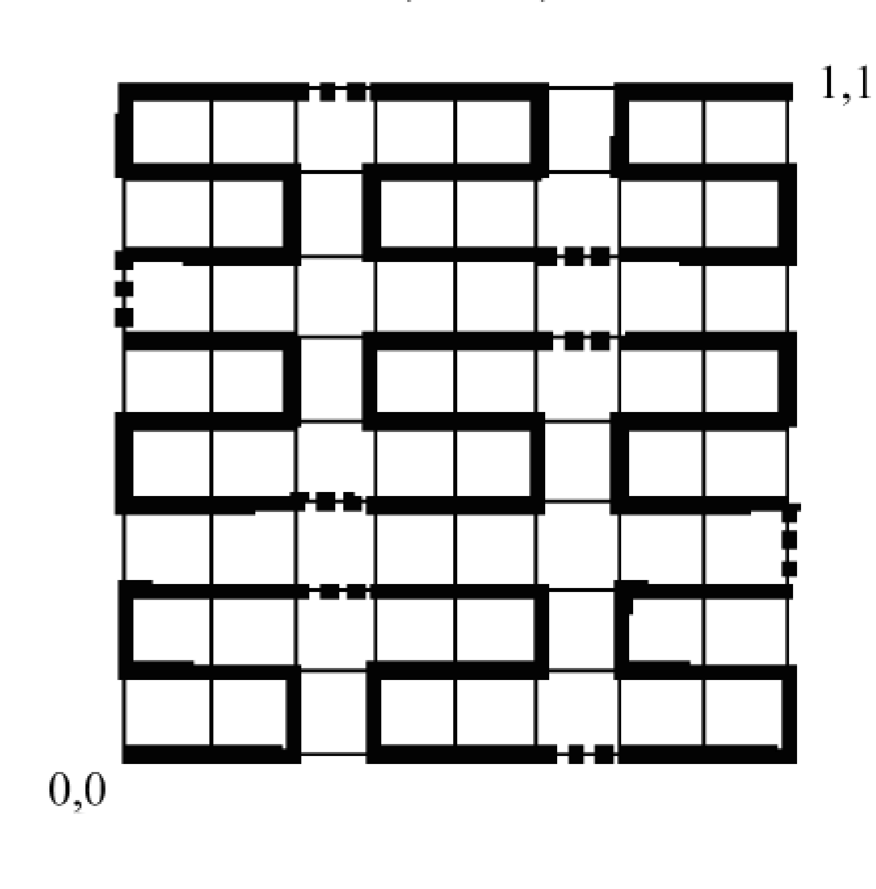


图 4.56



以(x,y)的形式给出两个 CPU 组件的坐标（$0\leqslant x, y\leqslant 1$），$x$ 是到左边的距离，y 是到顶端的距离。每个组件都是用直线连接到 SZ 曲线上离它最近的组件连接点，如果有多个，选择 x 和 y 坐标都更小的。两个组件之间的信号距离就是，组件到连接点的距离之和加上对应的组件连接点之间的 SZ 曲线长度。

给出 SZ 曲线的阶数（≤8），以及两个组件的实数坐标，计算输出二者之间的信号距离，保留小数点后 6 位。

滑雪（Skiing, World Finals 2014 - Ekaterinburg LA6779）

滑雪运动员在滑雪下山的比赛中要尽量多地经过一系列预定的地点，简单起见，我们把滑雪的路线简化成一个二维坐标系统，运动员从(0,0)出发，下山的方向就是 y 轴。假设运动员的垂直速度分量是常量 v_y。运动员可以改变速度的水平分量，但是受限于滑雪的装备，水平方向加速度的最大值是 a_{max}，开始时水平分速度是 0。

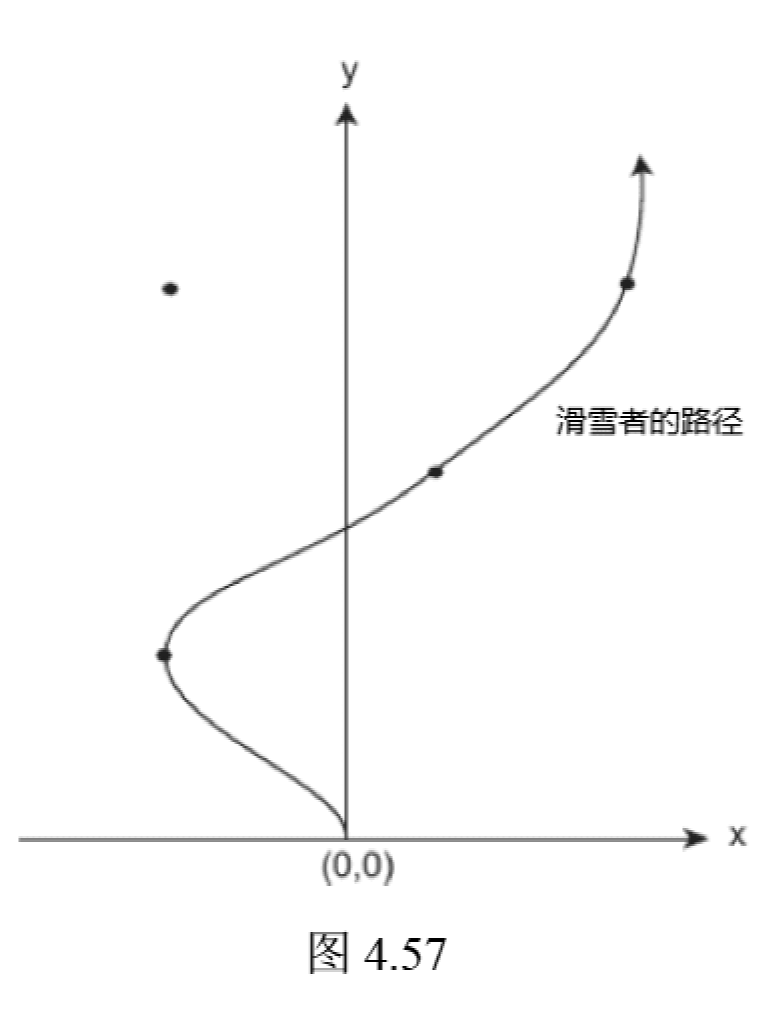


图 4.57

在图 4.57（对应于第一个样例输入）中，最优的路径刚经过了 4 个预定地点中的 3 个。如果 a_{max} 再小些，运动员可能就只能经过 2 个或更少的。

给出预定地点的个数 n（$0\leqslant n\leqslant 250$），$v_y$（$0\leqslant v_y\leqslant 10^5$），$a_{max}$（$0\leqslant a_{max}\leqslant 10^7$）。$v_y$ 的单位是 m/hr，a_{max} 的单位是 m/hr^2。然后依次给出编号为 $i = 1$～n 的要依次经过的地点的坐标(xi,yi)（$-10^5\leqslant xi, yi\leqslant 10^5$），以 m 为单位。

输出运动员可以最多一次性经过多少个预定的地点，并且依次输出经过的地点的编号。如果有多种答案，输出字典序最小的方案。如果一个也无法经过，输出“Cannot visit any targets”。可以认为 a_{max} 的扰动如果在 0.1 之内不影响最终答案。

午餐碟子（Buffed Buffet, Word Finals 2014 – Ekaterinburg, LA6771）

在一个餐馆有卖各种午餐碟子，可供任意挑选。有些碟子是固定菜量的，所以不能分开卖，只能卖整数个，称这些为“离散碟”。其他的因为是液体，所以可以选择任意的量，称为“连续碟”。

当然你更偏爱其中的某些碟子，但是这也跟你已经吃了多少有关。例如，即使是你爱饺子胜过土豆，但是如果已经吃了很多饺子，你会想吃些土豆。用如下的模型表示：每个碟子 i 都有一个初始的美味度 t_i，及其衰减率Δt_i。对于离散碟，你吃第 n 盘时感受到的美味度是$(n-1)\Delta t_i$。对于连续碟，在你已经吃过 x 克之后再吃另外无穷小的第 dx 克时感受到的美味度是$(ti - x\Delta ti)$dx。换句话说，你吃了 n 个离散碟或者 X 克的连续碟之后体验到的总的美味度分别是 $\sum_{n=1}^{N}(t_i-(n-1)\Delta t_i)$和$\int_0^X(t_i-x\Delta t_i)\mathrm{d}x$ 。

为简化起见，多种食物的美味度互相不影响。所以总的美味度就是各碟子的美味度之和。

给出碟子的数量 d（$1\leqslant d\leqslant 250$），以及你希望吃到的食物的总重量 w（$1\leqslant w\leqslant 10000$）。

接着 d 行以如下格式输出每个碟子的详细信息：

- ❑ $D\ w_i\ t_i\ \Delta t_i$ 表示一个单位重量 w_i 克的离散碟子，初始美味度是 t_i，衰减率是Δt_i。
- ❑ $C\ t_i\ \Delta t_i$ 表示一个连续碟子，初始美味度是 t_i，衰减率是Δt_i。

其中 w_i,t_i 和Δt_i 都是整数，并且满足 $1 \leqslant w_i \leqslant 10000$，$0 \leqslant t_i, \Delta t_i \leqslant 10000$。

输出在吃掉食物总重量为 w 克的前提下，能够品尝到的美味度之和的最大值。如果在给定的碟子中无法品尝到正好 w 克的食物，输出“impossible”。

监控（Surveillance, World Finals 2014 – Ekaterinburg, LA6780）

有一幢大楼，从上往下看是凸 n（$3 \leqslant n \leqslant 10^6$）边形。围绕它有 k（$1 \leqslant k \leqslant 10^6$）个地点可以安装摄像头。对于每个摄像头 i，输入两个整数 a_i,b_i（$1 \leqslant a_i,b_i \leqslant n$）。如果 $a_i \leqslant b_i$，表示摄像头可以覆盖每条符合 $a_i \leqslant j \leqslant b_i$ 的每条边 j；否则覆盖符合 $a_i \leqslant j \leqslant n$ 或者 $1 \leqslant j \leqslant b_i$ 的每条边 j。两个摄像头覆盖的区域可能会重叠。输出最少需要多少个摄像头才能覆盖所有的边。如果使用给出的摄像头无法全部覆盖，则输出“impossible”。

CCPC（中国大学生程序设计竞赛）

建塔游戏（Build Towers CCPC UVa12982）

有一种建塔游戏，如图 4.58 所示。有 n=8 个棍子。游戏一开始，每个棍子上有不同颜色的盘子。颜色共有 n−2 种。每种颜色的盘子都有 7 种大小的盘子各 1 个：0, 1, 2, 3, 4, 5, 6。所以总共有 7(n−2)个盘子。

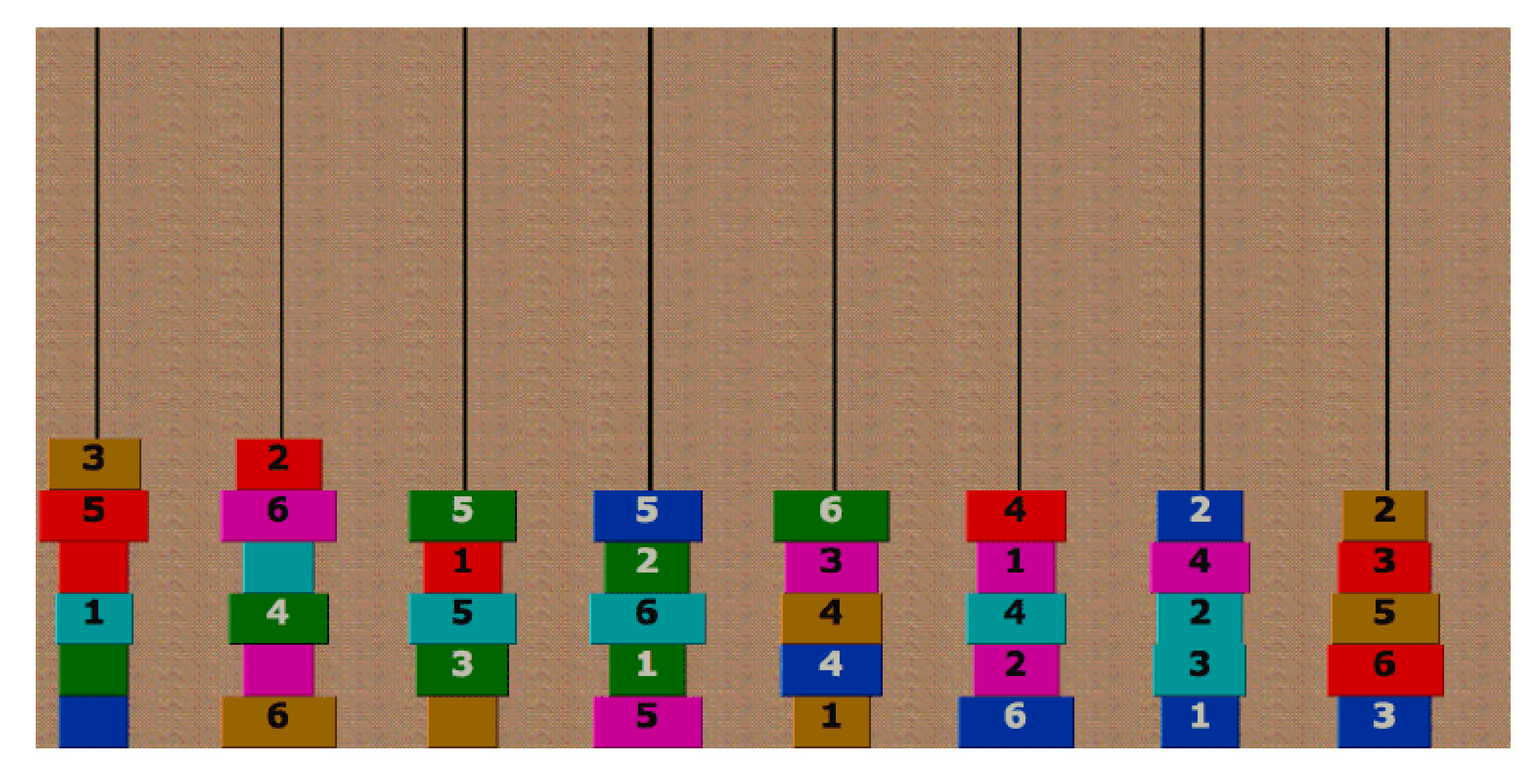


图 4.58

在游戏过程中，如果大小为 x 的盘子刚好放到同色 x+1 的盘子上边，它们就粘在一起不能再分开，接下来如果要移动就得一起移动。每次只能移动某个棍子顶端的盘子，有可能是几个盘子粘在一起移动。移动的目标棍子顶端必须是和被移动的盘子同色并且更大的盘子，或者目标的棍子是空的。

游戏的目标是要让 n−2 个棍子上，刚好每个都是 n−2 个塔，每个塔都是由从上到下大小刚好是编号 0,1,2,3,4,5,6 的 7 个盘子组成。给出初始每个棍子上放置的盘子的颜色，计算



达到这个目标所需的最小步数。

赤壁之战（The Battle of Chibi, CCPC, UVa12983）

黄盖在投降曹操时决定给他透露编号为 1～N 的 N（$1 \leqslant N \leqslant 10^3$）个真实的情报来博取信任。按照情报产生的时间排序，第 i 个情报的价值是 a_i（$1 \leqslant a_i \leqslant 10^9$）。黄盖决定在这些情报中选择长度为 M（$1 \leqslant M \leqslant N$）的价值严格递增的子序列。计算符合条件的选择方案的个数，输出其模 1000000007 的值。

在输入样例 1 中，黄盖要在 3 个情报中选择两个，他可以选择任意两个，因为所有的情报价值都是递增的。

在样例 2 中，黄盖没得选择，因为任意两个情报的价值都不是递增的。

样例输入：

```
2
3 2
1 2 3
3 2
3 2 1
```

样例输出：

```
Case #1: 3
Case #2: 0
```

选择金条（Pick the Sticks, CCPC 2015, UVa12984）

曹操要赏给杨修一些金条，每个金条可以认为是一条线段。其中有一条长度 L（$1 \leqslant L \leqslant 2000$）的线段作为容器，另外 N（$1 \leqslant N \leqslant 1000$）个作为货物，其中第 i 个长度为 a_i（$1 \leqslant a_i \leqslant 2000$），价值为 v_i（$1 \leqslant v_i \leqslant 10^9$）。要在这 N 个金条中挑选一些放到容器上，互相不能重叠。但是在容器的两端，被放置的金条可以突出容器边缘一部分，前提是重心不超过容器边缘。计算能在容器上放置的金条价值之和的最大值。

官渡之战（The Battle of Guandu, CCPC 2015,UVa12986）

曹操和袁绍的官渡之战中，有编号为 1～M 的 M 个不同的战场。附近有编号 1～N 的 N 个镇（$1 \leqslant N, M \leqslant 10^5$）。对于第 i 个镇，曹操可以招兵投入到战场 x_i。但是袁绍当时还太强大，村民为了自保必须给战场 y_i 送同样数量的人无偿加入袁绍的军队。对于第 i 个镇子，曹操招兵给每个人的报酬是 c_i（$0 \leqslant c_i \leqslant 10^5$）。加入袁绍军队的农民与曹操无关，无须给钱。一开始战场上双方都没兵。

对于曹操来说，不同的战场有不同的优先级，第 i 个战场的优先级是 w_i（$w_i \in \{0,1,2\}$），每个优先级的含义如下：

（1）w_i=2，非常重要，曹操要保证这些战场上他的兵力比对方多。

（2）w_i=1，要保证不比对方少。

（3）w_i=0，对于士兵数量没有任何限制。

这样他才有取胜的把握。计算曹操至少要投入多少钱才能保证获胜。如果问题无解，则输出“-1”。

游戏室（Game Rooms, CCPC 2015, UVa12991）

有一个包含编号为1～N的N（$2 \leqslant N \leqslant 4000$）层的大楼，其中第$i$层有$T_i$个乒乓球爱好者和$P_i$个撞球爱好者（$1 \leqslant T_i, P_i \leqslant 10^9$）。每一层要建个游戏室，但是具体是乒乓球还是撞球室未定。我们按照楼层定义一个游戏爱好者到他喜欢的游戏室的距离，记二者所在的楼层是a,b，那么距离就是$|a-b|$。

依次输入N和每个T_i, P_i。输出每个游戏爱好者到自己喜欢的游戏室距离之和最短的那个游戏室安排方案。

样例输入：

```
1
2
10 5
4 3
```

样例输出：

```
Case #1: 9
```

对于输入样例1，可以在第1层建乒乓球室，在第2层建撞球室。5个撞球爱好者需要上一层玩，4个乒乓球爱好者需要下一层玩。总的距离是9。

第 5 章　比赛难题选译

ACM/ICPC Europe – Central

山区风景（Mountainous landscape, Europe – Central 2014, LA6928）

在高低起伏的山区行走时，有山峰和低谷，但在某一点你希望知道你现在看到的是哪座山，如图 5.1 所示。

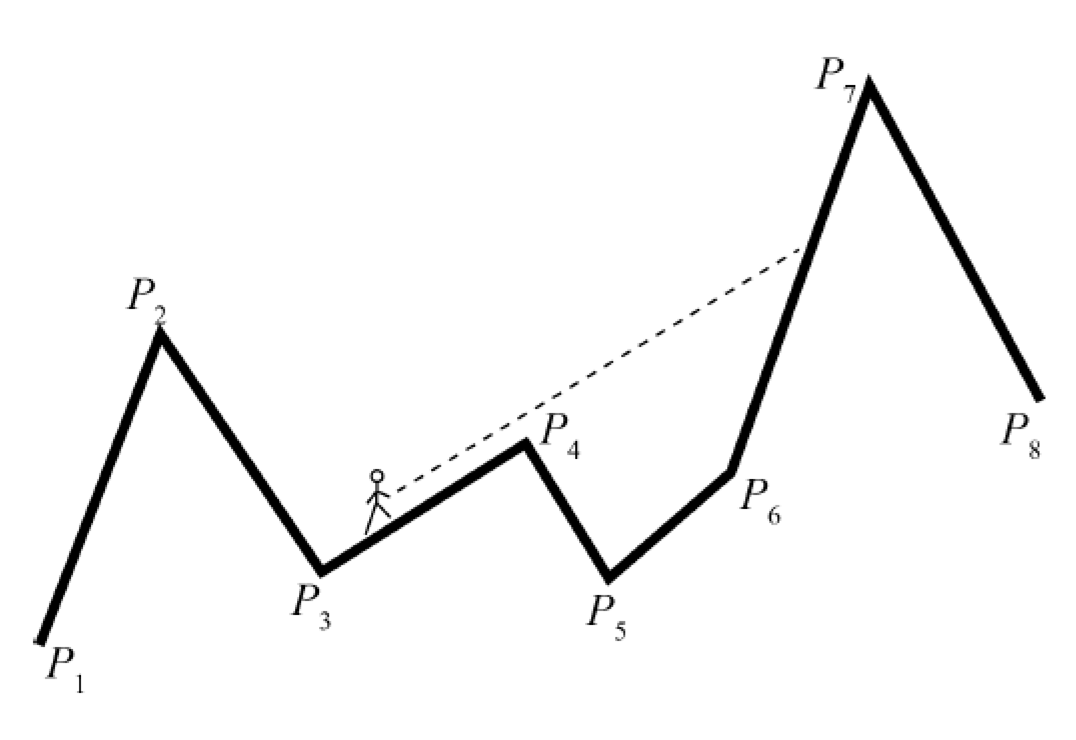


图 5.1

给出一个水平面上的多边形序列 $P_1P_2\cdots P_n$（$2\leqslant n\leqslant 100000$），以及每个点 i 的整数坐标 x_i,y_i（$0\leqslant x_1<x_2<\cdots<x_n\leqslant 10^9$，$0\leqslant y_i\leqslant 10^9$）。对于其上的每个线段 P_iP_{i+1}，找到最小的 $j>i$，使得线段 P_jP_{j+1} 的每个点都从 P_iP_{i+1} 可见（都严格位于射线 P_iP_{i+1} 上方）。问题无解则输出 0。

猪肉桶（Pork barrel, Europe – Central 2014, LA6936）

要建一个高速路网，有 n 个城市（$1\leqslant n\leqslant 1000$）和 m（$0\leqslant m\leqslant 100000$）条潜在的修路路径。每个路径，给出 x, y, w（$1\leqslant x\neq y\leqslant n$，$1\leqslant w\leqslant 1000000$）表示可以从城市 x 到 y 修一条费用为 w 的双向高速。一对城市之间可能有多种修路路径。但是要求每条高速的费用在 1 和 h 之间，同时要求被高速网连接起来的城市数量最大，需要计算出满足这些条件的路网的总费用最小值。

本题中，输入的 1 和 h 是 q 组（$1\leqslant q\leqslant 1000000$），首先给出最初的限制 l_1,h_1，然后第 j 组数据实际上是 l_j+c_{j-1} 和 h_j+c_{j-1}，其中 l_j 和 h_j 是要使用的限制，而 c_{j-1} 是对于 l_{j-1},h_{j-1} 计算出来的答案，其中 $1\leqslant l_j\leqslant h_j\leqslant 1000000$。

样例输入：

```
1
3
1 4 4
```

```
4 7 5
3 4 7
```

样例输出：

```
7
```

样例解释:

实际的限制条件 l,h 分别是 (1, 2), (1, 4), (2, 3), (3, 5) and (4, 5)。满足这些条件的最便宜的路网分别是：

{(1, 2), (4, 5)},

{(2, 1), (1, 5), (5, 4), (4, 3)}, {(1, 2), (1, 5), (3, 4)},

{(1, 5), (5, 2), (2, 3), (3, 4)}

{(3, 2), (2, 5), (1, 4)}.

ACM/ICPC Europe – Northeastern

隐藏的迷宫（Hidden Maze，Europe – Northeastern 2014, LA6946）

有一档名为“隐藏的迷宫”的真人秀节目。在这个节目中，两个参与者（往往是夫妻），要跑过一个由一些隧道连接起来的 n（$2 \leqslant n \leqslant 30000$）个大厅。每个隧道连接两个大厅，并且两大厅之间只能有一个隧道。

节目一开始，两个参与者被放到不同的大厅内。接着他们就需要尽快在规定的时间内会合。为了通过每个隧道，他们需要找到写在一小片纸上的正整数作为线索。

如果参与者能在规定时间内在某个隧道中会合，并且能在会合的隧道中找到线索，就算获胜。他们获得的奖励价值就是在对找到的所有线索值排序之后取的中间值。游戏规则保证了他们找到的线索数目都是奇数。

有两个玩家 Helen 和 Henry 注意到，每一期的节目中迷宫布局是不变的，并且把地图画出来了。然后又发现这个迷宫建造达到了这么一个效果：如果不重复走过一条隧道，那么每一对大厅之间就刚好只有一个路径。

迷宫是按照以下的随机算法建造的：

（1）选择大厅的数目 n，建造编号为 1～n 的 n 个大厅。

（2）在 1 到 n 之间，随机均匀地选择 i 和 j。

（3）如果 $i = j$，或者 i 和 j 已经连通，那么回到步骤（2）。

（4）在大厅 i 和 j 建一个隧道。如果所有隧道已经连通，算法退出。否则走到步骤（2）。

每个隧道只包含 1 个线索并且其数值 c（$1 \leqslant c \leqslant 10^6$）固定不变，但是用来生成其数值的算法未知。这个数值也已经标注在了地图上。

要通过 1 个隧道并且找到其中的线索，需要固定的 1 分钟时间。如果两人在最后的隧道中会合，进入隧道到跑到隧道中间需要半分钟的时间。给的时间仅仅够两个参与者使用



以下的最优策略来会合：

（1）他们使用最短路径会合。

（2）找线索时没有任何失误。

（3）不会走入不属于最短路径的隧道。

为了让参与者能够在某个隧道的中间会合，节目一开始时，他们就被放到一个这样的位置：所在的两个大厅之间的最短路径是奇数。

Helen 和 Henry 希望参加这个节目，他们已经把地图记下心里，并且他们非常有信心能够准时找到所有线索然后会合。一开始的两个大厅是随机均匀地从所有之间最短路径为奇数长度的大厅中选择。输入 n，然后是 n-1 行，每一行对应一个隧道，给出连接的大厅编号和其中的线索数值。

计算他们能获得奖励价值的期望值。要求误差在 10^{-9} 以内。

样例输入：

```
2
2 1 1
5
2 4 4
1 2 5
5 4 2
5 3 3
5
4 1 2
5 3 2
4 2 3
5 4 7
```

样例输出：

```
1
3.50
3.1666666667
```

样例解释：可以参考样例中输入的迷宫，圆的是大厅，双实线是隧道，方块中是线索的数值，如图 5.2 所示。

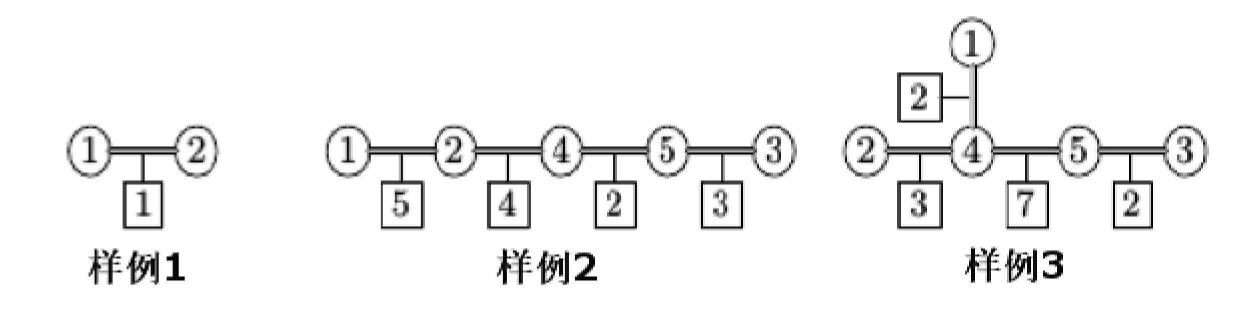


图 5.2

第 2 个迷宫中，可能有 6 对初始的大厅，如图 5.3 所示。决定最终奖励的线索用粗实线标出，期望值就是(4+5+2+3+4+3)/6 = 21/6 = 3.5。

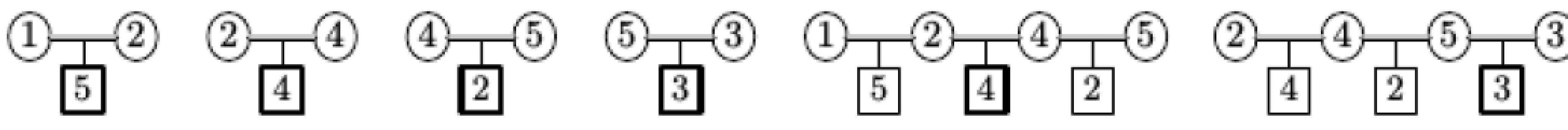


图 5.3

第 3 个迷宫中，6 个可能的初始大厅如图 5.4 所示，期望值就是(2+3+7+2+3+2)/6 = 19/6 = 3.16666666666…

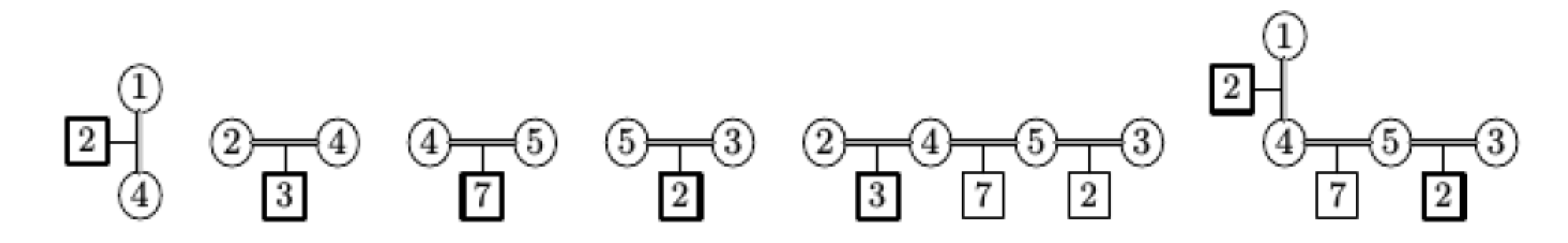


图 5.4

在这个迷宫中，如图 5.4 最右边所示，1、3 之间有两个为 2 的线索值。每一个都可以选择作为中间值，但是对奖品的价值没有影响。

五子棋（Gomoku, Europe – Northeastern 2014, LA6945）

本题是一个交互式问题。五子棋是一种两个玩家在二维 19×19 的网格上进行的游戏。每个格子要么是空的，或是黑方（玩家 1）的黑子，或是白方（玩家 2）的白子，但是只能有一个子。一开始网格是空的，两个玩家轮流移动，黑方先手。每一步一个玩家只可以将一个棋子放到一个空格子中。谁先把自己的 5 个棋子在同一条线连起来就获胜，可以是同一行，同一列，或者在和对角线平行的同一条直线上。

图 5.5 中，白方获胜。如果整个方阵都放满棋子且无人获胜，那么就是平手。

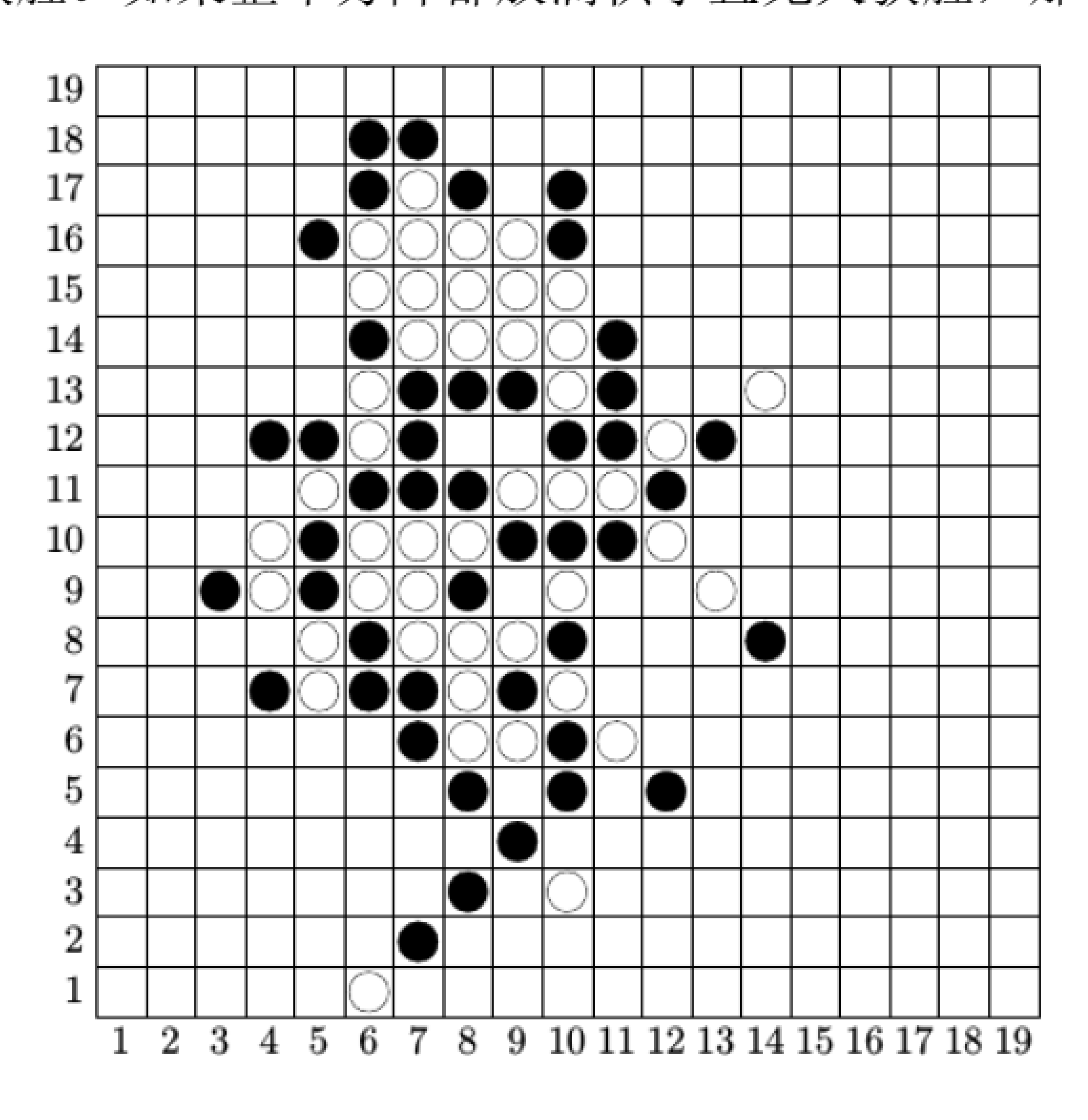


图 5.5

黑方采用如下的策略：第一步把棋子放到网格的中心，之后每一步放的位置都要尽量



使结果局面的得分最大化。

为了找到一个局面的得分，黑方考虑局面中所有可能形成胜利组合的位置。

（1）胜利组合就是棋盘上所有包含 5 个连续空格的水平或垂直或和对角线平行的直线。

（2）如果这个直线包含双方的棋子就要忽略掉。

（3）如果不包含任何棋子，也要忽略掉。

（4）如果这条直线包含 k（$1 \leqslant k \leqslant 5$）个黑子，并且没有白子，给这个局面加上 50^{2k-1} 分。

（5）对于有 k 个白子的直线，给这个局面减去 50^{2k} 分。

（6）最后，在局面得分上加一个 0 到 50^2-1 的随机数。

如果黑方的多种移动方式分数相同（非常罕见，因为上文加了随机数），就按照字典序选择落子坐标（x,y）最小的方案。

你的任务是写一个程序扮演白方并且打败黑方的这种策略，这个程序要跟使用上述策略的黑方玩 100 场游戏并且每场都胜，而且每次的随机数种子不同。

交互协议：每一步，你的程序需要做如下的动作。

（1）读入整数 x, y。

（2）如果 $x=y=-1$，程序退出。

（3）否则（x,y）就是黑方放子的位置（$1 \leqslant x, y \leqslant 19$）。

（4）打印白方每一步落子的坐标，注意写入输入缓存。

需要注意的是，五子棋的规则可能有许多变种，只考虑上文中所描述的规则。图 5.6 中给出样例输入对应的布局，只为了演示交互程序的输入格式，没有使用黑方的评估策略。

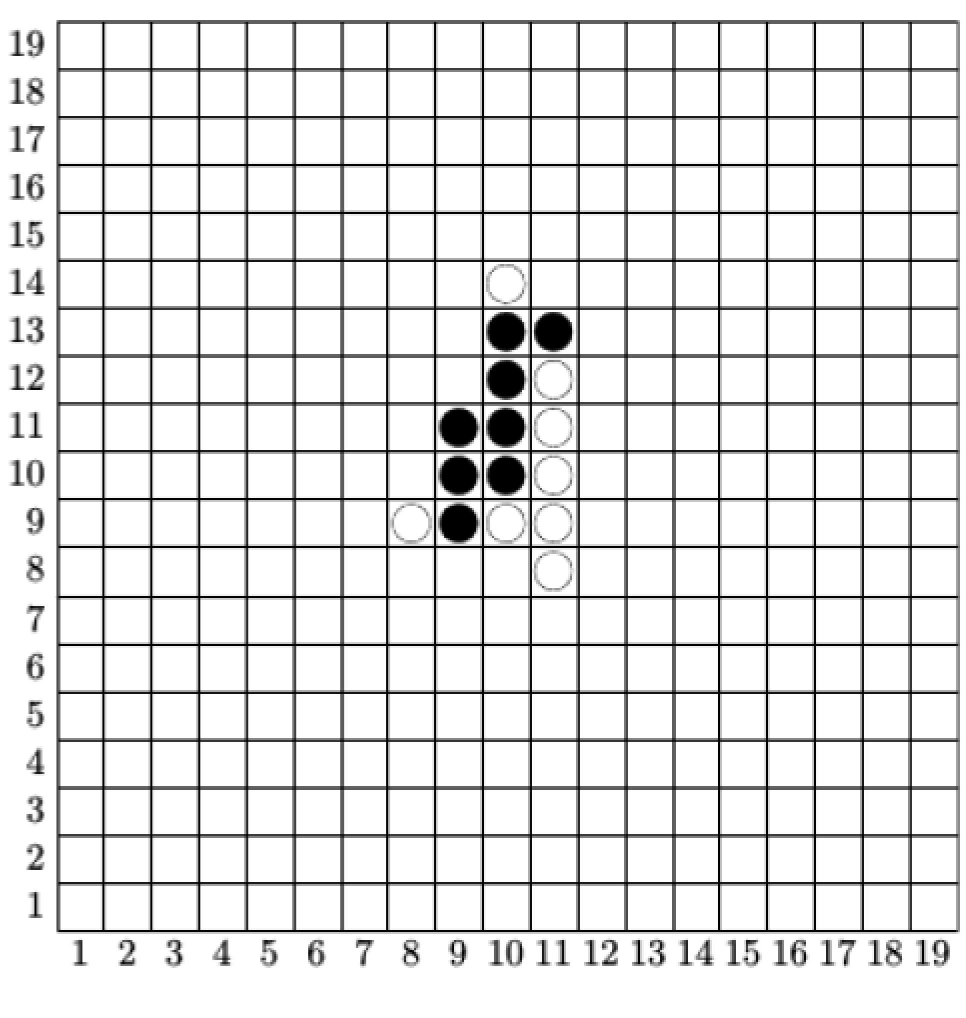


图 5.6

样例输入：

```
10 10
10 11
```

```
10 12
10 13
9 10
9 11
9 9
11 13
-1 -1
```

样例输出：

```
11 10
11 11
10 9
10 14
8 9
11 9
11 12
11 8
```

ACM/ICPC Asia – Taichung（台中）

世界图书日（World Book Day, Asia – Taichung2014, LA7010）

4 月 23 日是世界图书日，图书馆要举办一个活动来吸引读者参加。因为每种书图书馆只有 1 种，工作人员要求每个参与者写出希望借到的书的偏好列表，并且据此来做一些分配。每个参与者只能拿到偏好列表上的一本书；每本书也只能借给一个参与者。另外，图书馆对每个参与者评估出了一个权值。

在众多的图书分配方案中，对于其中的两个 S1 和 S2。有两种可能性让参与者 A 更喜欢 S1。一种是 A 能在 S1 中借到书，而在 S2 中不行；另外一种是 A 在 S1 和 S2 中都能借到书，但 S1 中借到的书在 A 的偏好列表的偏好值中比 S2 中的排得更靠前。

对于两种分配方案 S1 和 S2，如果更喜欢 S1 的所有读者的权重之和大于喜欢 S2 的读者的权重之和，我们说 S1 比 S2 更好，如果没有比 S1 更好的方案，就说 S1 是最优的。

例如，有 4 个参与者 A、B、C、D（权值分别是 5、5、2、2），编号 1～5 的 5 本书。每个作者的偏好列表（偏好值从左到右递减，括号中的表示偏好值并列）如下：

A (1 2), 5, 3

B (2 5 3), 1

C 2, 3, 4

D 1, 2

考虑如下 4 种分配模式，其中(A,1)表示 A 借到书 1：

S1：(A,1), (B,3), (C,2)

S2：(A,3), (B,5), (C,2), (D,1)

S3：(A,1), (B,2), (C,3)

S4：(A,2), (B,5), (C, 3), (D, 1)

对比 S1 和 S2，我们知道 A 和 D 更喜欢 S1，D 更喜欢 S2，B 和 C 没有偏好。因为 A 的权重是 5，D 的权重是 2，所以 S1 比 S2 更优。其他的方案依次进行比较之后发现没有比 S1 和 S4 更好的方案，所以他们都是最优的。

给出参与者的数量 m（$1 \leqslant m \leqslant 500$），以及编号为 1～$n$ 的书的数量 n（$1 \leqslant n \leqslant 1000$）。每个参与者的偏好列表中包含最少 1 个，不超过 100 本书。所有的参与者的不同权值 w 不超过 5 个，且有 $1 \leqslant w \leqslant 10$。计算是否有最优的分配方案，如果存在，输出其中能借到书的读者个数的最大值，否则输出“0”。

安全系统（Security System, Asia – Taichung 2014, LA7011）

给出一个非负整数 α 和长度为 n 的非负整数序列 $S=(s_1,s_2\dots s_n)$，其中（$0 \leqslant \alpha,s_i \leqslant 10000$，$1 \leqslant n \leqslant 15$），找到一个长度为 n 的最优非负实数序列 $T=(t_1,t_2,\dots,t_n)$，T 要满足以下两条属性：

（1）非递减（$t_1 \leqslant t_2 \leqslant \cdots \leqslant t_n$）。

（2）对于 $2 \leqslant i \leqslant n$，$t_{i-1}$ 和 t_i 必须满足（$t_i - t_{i-1} \leqslant \alpha$）。

记 R^n 为满足以上两属性的 T 组成的集合。为了确定 R^n 中的最优序列，定义 S 和 $T \in R^n$ 的距离为：$d(S,T)=\sum_{i=1}^{n}(t_i - s_i)^2$。最优的 T 就是和 S 距离最近的那个。可以假设最优的 T 中都是小于等于 10000 的有理数。

$n = 2$，$\alpha = 5$，S=(0,10)，很容易算出来最优的序列 T 是(5/2,15/2)。

考虑更复杂的情况 $n = 4$, $\alpha = 270$, S = (180, 450, 60, 980)，记 A = (307/3, 1112/3, 1369/3, 2099/3), B = (180, 1220/3, 1220/3, 2030/3), C = (1880/3, 690, 655, 900), D = (300, 620, 1677/2, 970)。

A 和 B 都是 R^4 的子集。但是因为 $c_3<c_2$ 并且 $d_2-d_1>270$，C 和 D 都不在集合 R^4 中。可以看到 $d(S, A)= 2231935/9>d(S, B)= 642200/3$。所以 B 优于 A。实际上 B 是最优的，因为它是 R^4 中和 S 距离最短的。

对于 $1 \leqslant k \leqslant n$，定义 f_k 如下，对于任何 $x \in [0,\ 10000]$，$f_k(x)$是$(s_1,s_2,\cdots,s_k)$到$(t_1,t_2,\cdots,t_k) \in R^k$ 的最短距离，其中 $t_k=x$。例如，$f_3(100.5)$就是 R^3 中任意的(s_1,s_2,s_3)到$(t_1,t_2,100.5)$的最短距离。注意 $f_1(x)=(x-s_1)^2$。根据定义，如果 f_n 已经计算出来，那么最短的距离 $d(S,T)$就等于 $f_n(x)$的最小值，其中 $0 \leqslant x \leqslant 10000$。而理论已经证明，$f_n$ 有如下性质：对于 $1 \leqslant k \leqslant n$，随着 x 的增长，$f_k(x)$递减到一个最小值然后再增长。给出 n 和 α 以及 $S=(s_1,s_2,\cdots,s_n)$，计算最短距离 d(S,T)，输出 a,b,c，其中 $a+b/c$ 就是 S 和最优的 T 的最短距离。

a,b,c 的具体含义如下：为了避免溢出，用一个 64 位整数组成 3 维元组(a,b,c)来表示一个有理数 r。其中，a 是其 r 整数部分，b/c 是 r 的有理数小数部分，其中 b,c 都是互素的非负整数。例如，$1 \leqslant k \leqslant n$，$r$=642200/3 那么就用(214066, 2, 3)来表示。另外，可以用(−3,1,2)来表示−2.5。注意如果 r 是整数，那么 b=0,c=1。

ACM/ICPC Asia – Daejeon

弹子游戏（Marbles, Asia – Daejeon 2014, LA6898）

有红蓝绿 3 种颜色数量分别为 a,b,c（$1 \leqslant a, b, c \leqslant 10000$）的弹珠扔到地上，每个球都给出坐标$(x,y)$，其中 $0 \leqslant x,y \leqslant 10000000$，并且没有两个球在同一个水平或者垂直直线上。位置都在水平面的第一象限，现在要画出 3 个边都和坐标轴平行且互相没有公共点的矩形 A,B,C。计算不同的画法中，矩形 A 中红珠，B 中蓝珠，C 中绿珠数量之和的最大值。

图 5.7 中给出一种珠子的分配情况，如果 3 个矩形如图 5.8 中分布，那么 A 中有 2 红珠，B 中有 2 蓝珠，C 中有 3 绿珠。那么总的可统计数量就是 7，实际上这就是能够得到的最大值。

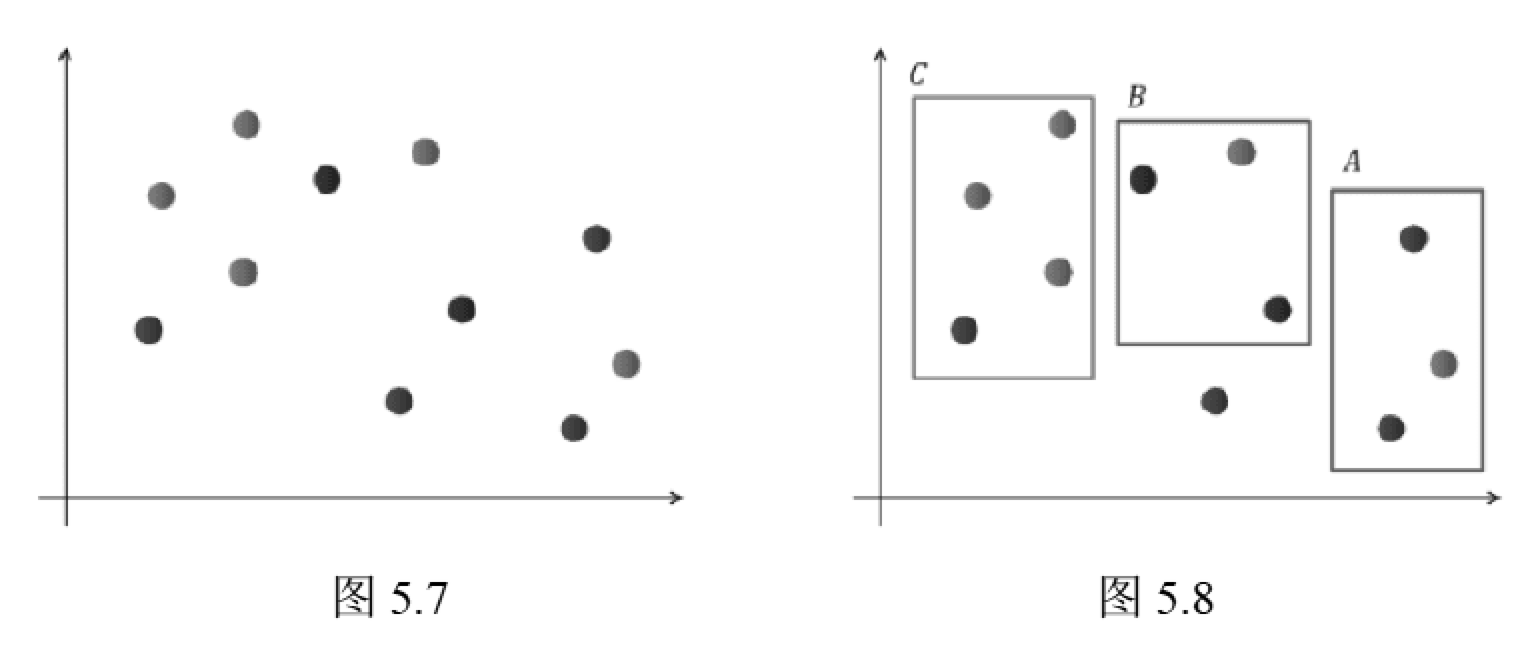


图 5.7　　　　图 5.8

ACM/ICPC Asia – Shanghai（上海）

矩阵革命（The Matrix Revolutions, Asia – Shanghai 2014, LA7138）

黑客帝国中，人类最后的城市锡安有 N（$2 \leqslant N \leqslant 50$）个独立的城堡，互相不连通，有 M（$0 \leqslant M \leqslant N^*(N-1)/2$）条可选的双向路径。尼奥可以选择建造一个虫洞或者道路来使两个城市连通，最多能建 K（$0 \leqslant K \leqslant M$ 且 $0 \leqslant K<N$）个虫洞。现在需要建最少的虫洞和道路使这些城堡全部连通。对于任意两个城堡，虫洞或者道路都能让其连通。给出所有的城堡以及其间能建道路的所有路径形成的图：每两点给出其能建的道路类型（只能建虫洞或道路或两者皆可）。计算能让尼奥实现他目标的建造方案个数，输出其模 10^9+7 的余数。

座位安排（Seat Arrangement, Asia – Shanghai 2014, LA7140）

Google 的办公室和普通的不一样，是一个 $n \times n \times n$ 的立方体布局，每一个格子都是 $1 \times 1 \times 1$ 的立方体，里面都有 1 个座位。定义位置(i, j, k)的座位是在第 i 层 j 行 k 列，都是从 1 开始计算。其中某些座位可能有人，给出所有有人的座位坐标。现在需要将所有的员工移到一起使得都全部连通，两个座位只有共享一个面的时候才算连通，共享一条边或者一个顶点不算。也就是坐标为(x_1,y_1,z_1)和(x_2,y_2,z_2)的两个位置在满足$|x_1-x_2|+|y_1-y_2|+|z_1-z_2|=1$ 时才算连通。每一次移动只能把 1 个人从一个座位移到相邻连通的座位。从座位 1 移到 2 时，2 不

必为空，但是整个移动过程完成之后，每个座位上只能有 1 个人。

计算一种移动方案使得移动完成之后所有有人的座位全部连通，并且移动次数不超过 200000。如果存在这种方案，首先输出移动次数，然后输出每一步移动的起始点和目的点坐标。如果有多重方案，任选一种输出。

游戏（Game, Asia – Shanghai 2014, LA7144）

S 和 E 一起玩一个游戏。一开始，每个人的攻击力是 H，能量 P 和防御力 D，都是整数。游戏采用回合制。每个回合，S 和 E 同时互相攻击一次。攻击之后每个人的攻击力减少$\lceil P_{对方}/D_{本方} \rceil$，其中$\lceil x \rceil$是不小于 x 的最小整数。

双方都有一个技能：切换当前的能量 P 和攻击力 H，每一回合开始之前可以选择是否使用这个技能，并且可使用任意多次。但是直到攻击开始之前，互相不知道对方的选择。

如果任何一方的攻击力下降到小于等于 0，就立刻死掉并且游戏结束。如果一方死掉，另外一方获胜。双方都死掉，游戏平局。胜者会获得 100 分，败者得−100 分，平局双方都得 0 分。双方都想尽量多的得分。

给出双方的 H,P,D（$1 \leqslant H,P,D \leqslant 50$），如果双方都使用最优的策略，S 得分的期望值 y 是多少？y 的误差不能超过 10^{-6}。

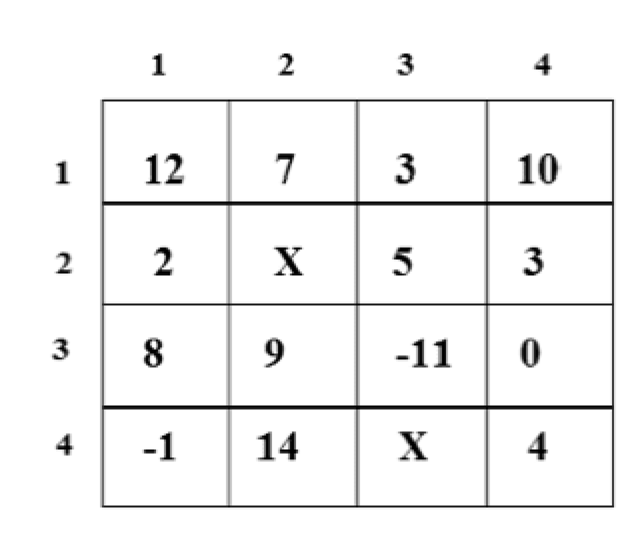
ACM/ICPC Asia – Dhaka（达卡）

网格之国的洪水（Flood in Gridland, Asia – Dhaka 2014, LA6920）

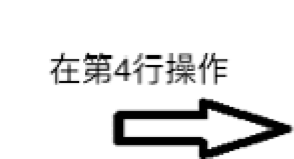
网格之国是 N 行×M 列（$1 \leqslant N,M \leqslant 75$）的一个网格，行列都是从 1 开始计算。第 i 行第 j 列表示为 land(i,j)，其高度为 H_{ij}（$-500 \leqslant H_{ij} \leqslant 500$）。如果 $H_{ij}>0$，表示为高地；如果 $H_{ij}<0$，则表示这里是海平面以下，或者说这个格子是无底深的海水。

现在政府希望通过以下两种操作使得每个格子的高度都变成在 L 和 U 之间（闭区间$[L,U]$内，$-1000 \leqslant L \leqslant U \leqslant 1000$），在此前提下要使得所有格子的高度之和最大化。

（1）让某一行所有格子的高度增加 1。如图 5.9 所示（其中 X 表示无底深的海水，所有操作对其无效）。



	1	2	3	4
1	12	7	3	10
2	2	X	5	3
3	8	9	-11	0
4	-1	14	X	4



12	7	3	10
2	X	5	3
8	9	-11	0
0	15	X	5

图 5.9

（2）让某一列所有格子的高度减 1，如图 5.10 所示。

	1	2	3	4
1	12	7	3	10
2	2	X	5	3
3	8	9	-11	0
4	0	15	X	5

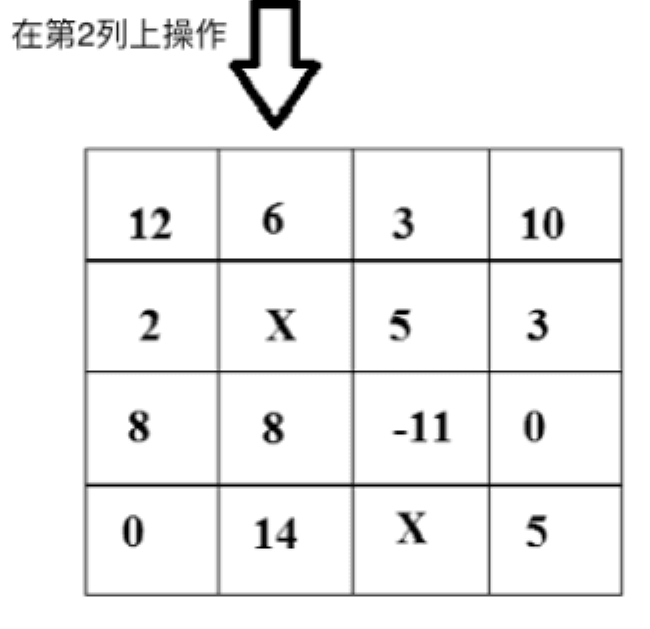


图 5.10

在 90%的测试数据中，$N,M \leqslant 20$。

如果问题无解，输出“Impossible”；否则输出所有格子的高度之和。接下来输出 N 个非负整数 R_i（$R_i \leqslant 1000000$），其中 R_i 表示应用在第 i 行的操作 1 的个数。再输出 M 个非负整数 C_j（$C_j \leqslant 1000000$），其中 C_j 是应用在第 j 列的操作 2 的次数。

ACM/ICPC Asia – Mudanjiang（牡丹江）

卡片游戏（Card Game, Asia – Mudanjiang 2014, LA6971）

Earthstone 是一种两个玩家玩的卡片收集游戏，回合制比赛形式进行。玩家一开始有一些基础的卡片，但是同时可以通过购买附加卡包来获得一些更罕见更强的卡片，也可以通过比赛赢取。卡包要通过游戏中的金币购买，而金币可以通过完成随机的日常任务，或者比赛获胜，或者通过现实货币购买而获得。

游戏中的每场战斗都是回合制的二人比赛。轮到一个玩家是，他可以选择打出任意的牌作为奴才，并且命令这些奴才攻击对方，每张牌有两个基本属性：

（1）攻击力 A_i，如果奴才攻击另外一个角色或者被攻击了，对被攻击者会造成 A_i 的伤害。一个角色如果攻击力小于等于 0 就不能发起攻击。

（2）生命值 H_i，一开始奴才有 H_i 点生命值。被攻击之后，就会减去对应的伤害值。如果小于等于 0，奴才就会被杀掉然后抛弃。

如果奴才攻击另外一个奴才，双方会同时受到伤害。除了奴才，每个玩家都有一个包含某种初始生命值的英雄，英雄攻击力为 0。如果英雄被杀，玩家就输了。游戏中的角色指的是英雄或奴才。除了两个基本属性，奴才包含以下技能。

（1）Charge-前冲：奴才在被召唤的这一轮，不能发起攻击，除非使用前冲技能。

（2）DivineShield-神盾：吸收第一次被攻击时的非零伤害，然后就失效。

（3）Taunt-嘲笑：对方在攻击无此技能的角色之前，必须攻击有此功能的奴才。

（4）Windfury-风之怒：在轮到一个玩家时，他最多可以发一次命令，让所有现有奴才或者新召唤的有前冲技能的奴才发起一次攻击。但是如果某个奴才有风之怒技能，它可以连发两次这样的命令。

Edward 在游戏中买了 400 个卡包，然后觉得应该无敌了，所以他想开单挑一下。现在轮到他了。桌上已经有 $X+Y$ 个奴才，X 个是爱德华这边的，其他是敌方的。爱德华手上还有 Z 张卡片。英雄的生命是 M。一轮中打出的牌数没有限制，桌上的奴才数量也没有限制。

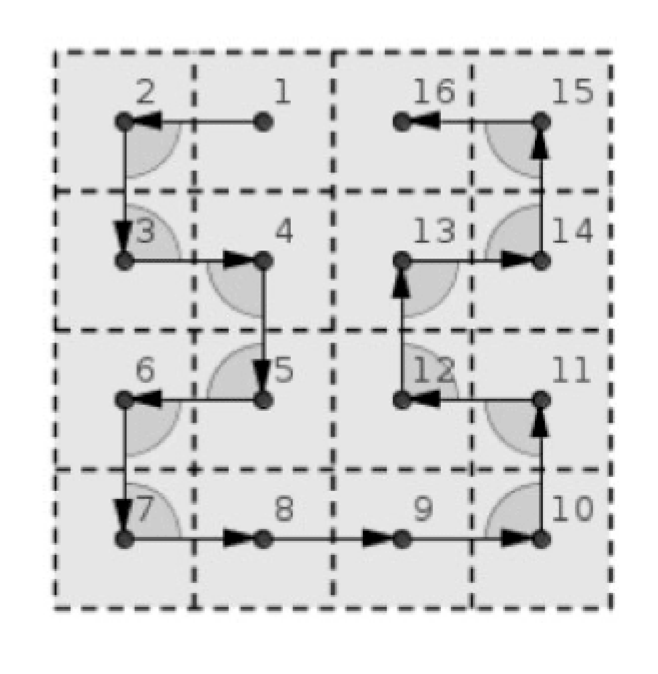
Edward 希望你找到一个长期的游戏策略。希望在下一轮轮到对手时减少他收到的潜在伤害。这个伤害使用对手奴才的攻击力之和来计算，如果某个奴才有 Windfury 技能，结果还要乘 2。如果有多解，就使用能够对对方英雄造成最大伤害的那个。但是如果存在一个策略能够在当前一轮获胜，就使用这种策略。

输入 X,Y,Z（$0\leqslant X+Z\leqslant 8$，$0\leqslant Y\leqslant 15$）和 M（$1\leqslant M\leqslant 100$），然后以“$A_i/H_i$能力”的格式输入，分别输入 X,Y,Z 三部分对应的奴才或卡片的能力。“能力”部分包含 0 或多个技能名称。输出下一轮收到的最小的潜在伤害以及能对对方英雄造成的最大伤害。如果对方英雄在当前这一轮杀不掉，就输出“Well played”代替。

挖掘机考试（Excavator Contest, Asia – Mudanjiang 2014, LA6973）

蓝翔技校举办一场挖掘机比赛。参赛者需驾驶挖掘机通过一个 $N\times N$（$2\leqslant N\leqslant 512$）网格形状的场地，把场地边缘的两个不同的格子作为起点和终点把每个格子刚好走一次。还需要至少进行 $N\times(N-1)-1$ 次转弯，每次转弯只能是向左或者向右转 90°。请为参赛者规划出一个可行的路径，如果问题有多解，输出任意一个即可。

提示，N=3,4 时的路线如图 5.11 所示。



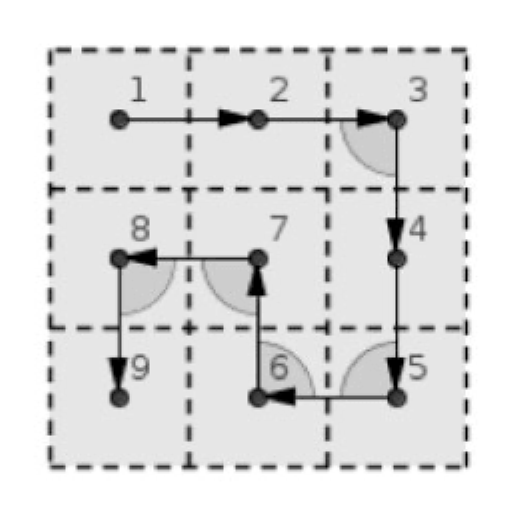


图 5.11

花园和洒水（Garden and Sprinklers, Asia – Mudanjiang 2014, LA6975）

大学里面有一个花园，圆心是(X_0,Y_0)，半径 R（$1\leqslant R\leqslant 10^8$）。在$(X_1,Y_1)$、$(X_2,Y_2)$两点（不重合）上已经有两个洒水器，现在需要为第 3 个洒水器选址，满足以下条件：

（1）3 个洒水器不共线。

（2）第 3 个应该在花园的边上或者花园外。

（3）洒水器的坐标都必须是整数。

（4）3 个洒水器形成的三角形的面积应该等于 $2S$（$1\leqslant S\leqslant 10^8$）。

依次输入整数 S, X_0, Y_0, R 以及 X_1, Y_1, X_2, Y_2。计算满足以上条件的第 3 个洒水器的位置个数。输入的所有坐标的绝对值不超过 10^8。

样例输入：

```
1
4
0 0 4
-1 0 1 0
```

样例输出：

```
14
```

样例输入中，所有合法的位置有 (−3, 2), (−2, 2), (−1, 2), (0, 2), (1, 2), (2, 2), (3, 2), (−3, −2), (−2, −2), (−1, −2), (0, −2), (1, −2), (2, −2), (3, −2)。

雅可比模式（Jacobi Pattern, Asia – Mudanjiang 2014, LA6978）

如果两个循环等价的序列连在一起，就形成一个雅可比模式。例如，把{1, 2, 3, 4}和{3, 4, 1, 2}串接起来，得到{1, 2, 3, 4, 3, 4, 1, 2}。现在给出一个序列$\{A_1, A_2, \cdots, A_N\}$，序列中包含 M 种不同的数字（$1 \leqslant N, M \leqslant 5000$，$1 \leqslant A_i \leqslant M$），计算它有多少个连续子序列能形成雅可比模式。

输出这种子序列的个数 X，及其不同长度的个数 Y。按照长度的递增序，对每种长度输出 L_i 和 C_i，前者是长度，后者是对应的序列个数。按照递增序输出 C_i 个数字 $P_{i1}, P_{i2}, \cdots, P_{iCi}$，其中 P_{ij} 表示从 P_{ij} 开始有一个长度为 L_i 的雅可比模式子序列。

☞注意：

> 两个序列，如果其中一个能把它的某个后缀从后面移到前面得到另外一个序列，那么这两个序列就是循环等价的。{1, 2, 1, 2, 2, 1} 和 {1, 2, 2, 1, 1, 2}是循环等价的。但{1, 2, 1, 2, 1} 和{1, 1, 2, 2, 1}不是。每个序列和自身都是循环等价的。

样例输入：

```
2
12 4
1 1 1 2 3 4 3 4 1 2 2 1
3 4
1 2 4
```

样例输出：

```
6 3
2 3 1 2 10
4 2 5 9
8 1 3
0 0
```

提示：

> 在第 1 个样例中，有 3 个长度为 2 的雅可比模式序列：{1, 1}, {1, 1}, {2, 2}，2 个长度为 4 的：{3, 4, 3, 4}, {1, 2, 2, 1}。长度为 8 的只有 1 个：{1, 2, 3, 4, 3, 4, 1, 2}。在第 2 个样例中，没有雅可比模式。



ACM/ICPC Asia – Tehran（德黑兰）

线性拟合（Line Fiting, Asia – Tehran 2014, LA7022）

线性函数拟合是要针对一个未知函数 F：ℝ→：ℝ 形成的 n（$1\leqslant n\leqslant 10^5$）个样本点来找到一个能够最好的匹配这些点的线性函数 $\overline{F}$ 。

但每一个样本点 x_i 对应的 $F(x_i)$是按照离散概率分布的方式给出：给出一个离散集合 $y_{i,1}\cdots y_{i,mi}$（$1\leqslant m_i\leqslant 10$，$0\leqslant x_i\leqslant 10^9$），每个 $y_{i,j}$ 都有一个概率：$Pr[F(x_i)=y_{i,j}]=p_{i,j}/100$，其中 p 是不大于 100 的非负整数。使用期望值的概念定义 $\overline{F}$ 的误差值：$error(F,\overline{F})=\max_{1\leqslant i\leqslant n}\left\{E\left[\left|(F(x_i)-\overline{F(x_i)})\right|\right]\right\}$。

要找到误差值最小的线性函数 $\overline{F}=ax+b$，输入所有的样本点及其概率。计算 $\overline{F}$ 误差值的最小值，保留小数点后 1 位。

ACM/ICPC Asia – Xian（西安）

无限电池厂（Unlimited Battery Works, Asia – Xian 2014, LA7044）

小苹果发明了一个在有根树上玩的棋类游戏。树上有编号 1～n 的 n 个顶点（$1\leqslant n\leqslant 50$），每个顶点给出一个整数 A_i（$1\leqslant A_i<50$），及其父顶点编号。开始每个顶点上都有一颗黑色的棋子。每一步，你可以随机选择一个顶点 i，不管这个顶点的棋子是什么颜色，都把它转换成白色。同时，如果 j 在 i 的子树内且 i 到 j 的最短路径上的边数不超过 A_i，每个顶点 j 上的结点也会变成白色。假如每一步都均匀随机选择树上的一个顶点，计算要把整棵树变成白色所需要的步骤数的数学期望。

注意：

如果树上所有顶点都是白色，就不能再进行任何转换。

ACM/ICPC Asia – Anshan

随机翻转游戏机（Random Inversion Machine, Asia – Anshan 2014, LA7051）

有一个游戏机，包含排成一行的编号为 1～$2n$ 的 $2n$ 个插槽（$1\leqslant n\leqslant 2000$），可以玩一种多轮游戏。每一轮可以把插槽分成 k（$1\leqslant k\leqslant n$）段，相邻的两段之间有标记。每一段必须包含偶数个插槽。接下来游戏机产生$\{1,2\cdots 2n\}$的一个随机排列并且在插槽上显示这些排列，最后计算所有段的逆序数对个数的乘积作为这一轮的得分。

可以玩任意多轮，但是每一轮的分段方式必须完全不同。也就是说，不能有两轮在同一个位置都包含分段标记。总的得分是每一轮得分的总和。但是现在机器中毒了，每次产生一个随机排列之后会对这个排列中偶数位置的数字进行排序。

例如，n=2, k=1，生成的排列是(2,4,1,3)。病毒就会对在插槽 2 和 4 中的数字 4,3 进行排序，把序列变成(2,3,1,4)。所以这一轮的得分是 2（逆序数对是 (2, 1) 和(3, 1)）。现在需要计算游戏最高得分的期望值。

因为问题的答案可能非常大，所以把它变成不可约分数 A/B 的形式，并且输出($A*B^{-1}$) mod (10^9+7)的形式。这里 B^{-1} 是 B 模 10^9+7 的逆。输入保证 B 和 10^9+7 互素。

样例输入：

```
3
1 1
2 2
2 1
```

样例输出：

```
500000004
250000002
166666670
```

提示：

对于输入样例，最高得分的期望值分别是 1/2, 1/4, 13/6。

Trie 树（Trie, Asia – Anshan 2014, LA7057）

给出一个包含编号为 1～n 的 n（$1 \leqslant n \leqslant 10^5$）个结点，根结点编号为 1 的 Trie 树，每条边上有个小写字母。

定义 S_i 是从根到结点 i 的路径上的字符连接成的字符串，显然 S_1 是空串。对于集合 $P \subseteq \{1,2,\cdots,n\}$，$S_P=\{S_i|i \in P\}$。对这个 Trie 定义一个用 P 来描述的特征：我们说一个字符串 a 拥有此特征，当且仅当存在一个字符串 $b \in S_P$ 且 a 以 b 为后缀，其中 a 以 b 为后缀的意思是 a 的最后$|b|$个字符刚好是 b。需要注意的是，每个字符串都以空串为后缀。

定义作用于 P 的映射 f: $f(P)$也是$\{1,2,\cdots,n\}$的一个子集，其中 $i \in f(P)$当且仅当 S_i 是 S_P 中某个 S_j 的后缀。记 D_i 为 S_i 当前拥有的特征个数。需要注意的是，当新的特征添加时，D_i 可能会改变。

给出 Trie 的结构，接着给出 m（$m \leqslant 10^5$）个操作，每个操作给出其类型和集合 P。类型 1 表示要增加一个特征 P；类型 2 表示要计算 $\sum_{i \in f(p)} D_i$。针对每个类型 2 的计算，输出其结果。输入输出格式请参考原题。

样例说明：

输入样例对应的 Trie 树结构如图 5.12 所示，其中：S_1=“”，

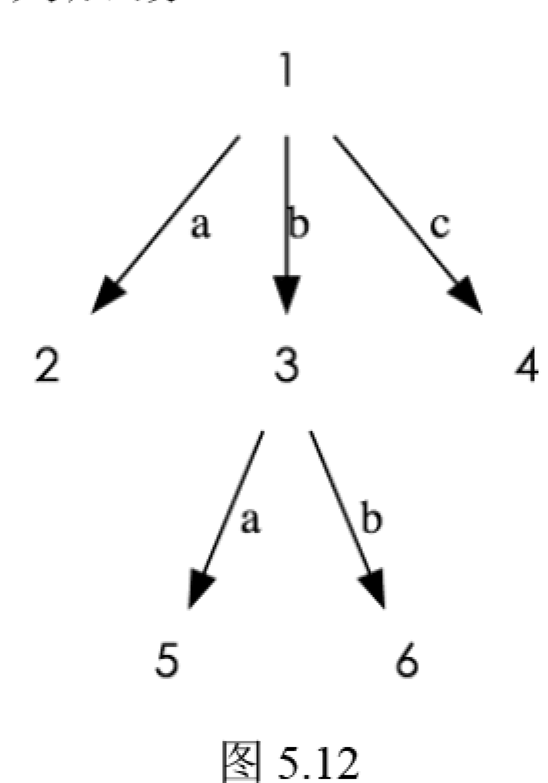


图 5.12

S_2=“a”，S_3=“b”，S_4=“c”，S_5=“ba”，S_6=“bb”。

输入的操作执行过程如下：

（1）1 2 3 4，增加一个特征集合 P={3,4}，其 S_P={“b”,“c”}，拥有特征 P 的字符串有{S_3=“b”, S_4=“c”, S_6=“bb”}，此时 D_1=0,D_2=0, D_3=1, D_4=1,D_5=0, D_6=1。

（2）2 2 5 6，P={5,6}，其 S_P={“ba”,“bb”}，$f(P)$ = {1,2,3,5,6}。计算结果是 D_1+D_2+D_3+D_5+D_6=2。

（3）1 2 2 3，P={2,3}，其 S_P={“a”,“b”}，拥有此特征的字符串有{S_2=“a”, S_3=“b”, S_5=“ba”, S_6=“bb”}，此时 D_1=0, D_2=1, D_3=2, D_4=1,D_5=1, D_6=2。

（4）2 2 4 5，P={4,5}，S_P={“c”,“ba”}，$f(P)$={1,2,4,5}，计算结果为 3。

（5）2 1 6, P = {6}, S_P={“bb”}，$f(P)$={1,3,6}，计算结果为 4。

样例输入：

```
1
6
1 a
1 b
1 c
3 a
3 b
5
1 2 3 4
2 2 5 6
1 2 2 3
2 2 4 5
2 1 6
```

样例输出：

```
2
3
4
```

ACM/ICPC Asia – Beijing（北京）

重新搞一次 GRE 单词（GRE Words Once More!, Asia – Beijing 2014, LA7064）

现在的 GRE 用一个由有向无环图（DAG）构成的状态机来记录单词，图上有编号为 1～N 的 N 个顶点和 M 条边（$2 \leqslant N \leqslant 10^5$，$0 \leqslant M \leqslant 10^5$），每条边都标有 1 个整数，某些顶点标记为特殊。一个 GRE 单词是通过把从顶点 1 到某个特殊顶点的路径上的标记串接起来获得。

给出 Q（$1 \leqslant Q \leqslant 10^5$）个问题，其中第 i 个是求按照字典序第 k_i（$1 \leqslant k_i \leqslant 10^8$）小的 GRE 单词的长度，如果单词不存在，直接输出“-1”。

样例输入：

```
1
3 3 4
1 1
1 2 1
1 3 12
2 3 3
1
2
3
4
```

样例输出：

```
Case #1:
1
2
1
-1
```

提示：

样例输入中总共有 3 个 GRE 单词，按照字典序给出分别是：
（1）（1）。
（2）（1, 3）。
（3）（12）。

只是一个错误（Just A Mistake Asia – Beijing 2014, LA7067）

给出一棵包含编号为 1～N 的 N（$1 \leqslant N \leqslant 200$）个顶点的树，及其每一条边 u,v（$1 \leqslant u, v \leqslant N$）。随机均匀地选择{1,2,3…$N$}的一个排列 $p_1, p_2, \cdots, p_N$，然后执行如下步骤：

（1）令集合 $S=\emptyset$，依次考虑顶点 $p_1, p_2, \cdots, p_N$。

（2）对于 p_i，如果 S 中没有 p_i 相邻的顶点，就把 p_i 加入 S。

计算 S 大小的期望值 SN，并且输出 SN×N! mod (10^9+7)。

样例输入：

```
2
4
1 2
1 3
1 4
3
1 2
2 3
```

样例输出：

```
Case #1: 60
Case #2: 10
```

提示：

第 1 个输入样例中，有 4 个顶点，所以有 4!种排列。排列 1 2 3 4，最终集合中就是顶点 1。排列 2 1 3 4，最终集合就是 2,3,4。很显然，如果排列中第 1 个元素不是 1，会得到一个大小为 3 的集合。否则，会得到一个大小为 1 的集合。因为有 18 个第 1 个元素不是 1 的排列，答案就是$(3\times 18+1\times 6) \bmod (10^9+7)=60$。

ACM/ICPC Asia – Guangzhou（广州）

雍正之死（Yong Zheng's Death, Asia – Guangzhou 2014, LA7071）

历史学家在故宫中发现了揭示雍正死因的加密文档，文档中共有 n（$1\leqslant n\leqslant 10000$）个小写字母字符串组成一个集合 $S=\{s_1,s_2,\cdots,s_n\}$，s_i 长度 L 符合 $1\leqslant L\leqslant 30$。从这个集合中可以创造一些死亡密码。一个字符串称为死亡密码当且仅当它能被分成两个子串 u 和 v，其中 u 和 v 都是 S 中某个字符串的非空前缀，它们可以是同一个字符串或者是不同的字符串的前缀。给出 S，输出其中死亡密码的个数。

样例输入：

```
2
ab
ac
0
```

样例输出：

```
9
```

提示：

对于输入样例，所有的死亡密码是{aa, aba, aca, aab, aac, abac, acab, abab, acac}。

平方后的频率（Squared Frequency, Asia – Guangzhou 2014, LA7075）

给出一个十进制的，四舍五入到小数点后 K（$K\leqslant 9$）位的数字 F（$0<F<1$）。现在需要求出一个分数 P/Q（P 和 Q 都是正整数），使得$(P/Q)^2$四舍五入到小数点后 K 位之后刚好等于 F，并且使得 Q 最小化。分别输出 P 和 Q，如果问题有多解，输出 P 最小的那个。

ACM/ICPC Asia – Tokyo（东京）

展览会（Exhibition, Asia – Tokyo 2014, LA6841）

市政府要从 n（$1\leqslant n\leqslant 50$）个产品中挑选 k（$1\leqslant k\leqslant n$）个来参加展会。其中第 i 个产品有 3 个参数，分别是 x_i、y_i 和 z_i（$1\leqslant x_i,y_i,z_i\leqslant 100$）。选择的 k 个产品 $i_1,\cdots,i_k$（$1\leqslant i_j\leqslant n$）必须使以下的表达式值最小：$e=\left(\sum_{j=1}^{k}x_{i_j}\right)\left(\sum_{j=1}^{k}y_{i_j}\right)\left(\sum_{j=1}^{k}z_{i_j}\right)$ 如果有多种选择方案，会随机选择一个。

你为制造产品 1 的公司工作。给出 3 个参数 A、B、C（$1\leqslant A, B, C\leqslant 100$），意思是如果要将产品的参数分别减少至$(1-\alpha)x_1, (1-\beta)y_1, (1-\gamma)z_1$（$0\leqslant\alpha,\beta,\gamma\leqslant 1$），就要投入 $\alpha A+\beta B+\gamma Cd$ 的预算。计算要让产品 1 可能被政府选择，需要的最小预算。可以假设其他公司产品的参数不变。输出误差应该在 10^{-4} 以内。

L∞跳跃（L∞ Jumps, Asia – Tokyo 2014, LA6842）

对于平面上的任意两个点(p,q)和$(p`,q`)$，定义 L∞距离为 $\max(|p-q|, |p`-q`|)$。假设一开始你站在(0,0)，并且需要移动到(s,t)($|s|,|t|\leqslant n*d$)。为此需要跳跃恰好 n 次，每次跳跃必须是跳 d（$1\leqslant d\leqslant 10^{10}$）个 L∞距离，且必须跳到一个整数坐标点。另外如果没跳够 n 次。即使到了(s,t)也不能停下。

每一次跳跃都有费用。给出另外 $2n$（$1\leqslant n\leqslant 40$）个整数 $x_1,y_1,x_2,y_2,\cdots,x_n,y_n$，都满足 $\max(|x_i|,|y_i|)=d$。第 i（从 1 开始）次跳跃的费用定义如下：假设这次跳跃之前的位置是（p,q），考虑你能跳跃到的所有整数点集合，这个集合包含一个特定矩形边上的所有整数点。把整数 1 分配给点$(p+x_i,q+y_i)$，这个点和 q 的距离刚好就是 d*L∞，分配 $2,3,\cdots,8d$ 给剩下的点（按照逆时针顺序分配，这里假设正 x 轴是右，正 y 轴是上）。分配的整数就表示跳到这些点所需要的费用。

举例来说，图 5.13 展示了在第 i 次跳跃时，当前位置是(3,1)，且 d=2。这些数字表示了 x_i=−1 和 y_i=−2 开始的所有费用。

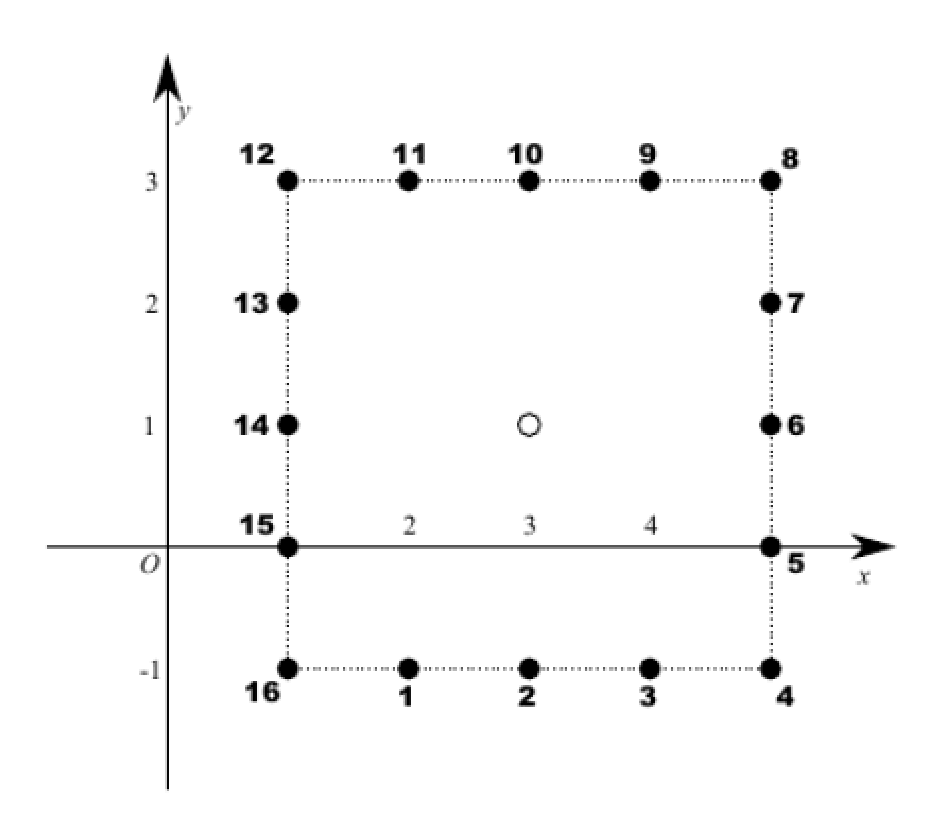


图 5.13

计算要跳跃到指定的目标所需要的费用之和的最小值。

ACM/ICPC Asia – Bangkok（曼谷）

最优降落地点（Optimal Landing Location, Asia – Bangkok 2014, LA6846）

岛上有 3 个空仓库，随处可降落飞机。需要为一个货机选择降落地点，记最优地点为 L，选址的规则如下：

（1）仓库容量和飞机上的货物数量，都以货物的个数为单位。

（2）从 L 出发，需要 3 辆不同的卡车在 L 和 3 个仓库之间运送货物。卡车会从 L 出发，沿直线到达仓库，卸货后原路返回。

（3）卡车可以往返多次把货物运到仓库，但是运完所有货物后必须返回 L。

（4）如果 3 个仓库的容量之和大于飞机上运送的货物数量，可以分别决定每个仓库要存储的货物数量。

（5）卡车的容量都是 1，一趟只能拉 1 个单位的货物。

给出 3 个仓库的位置(A_x, A_y)、(B_x, B_y)和(C_x, C_y)（$0 \leqslant A_x, A_y, B_x, B_y, C_x, C_y \leqslant 1000$），三者容量以及飞机上的货物数量分别是 C_A, C_B, C_C 和 W_P（$20 \leqslant 2C_A, 2C_B, 2C_C, W_P \leqslant 2000$，$W_P \leqslant C_A+C_B+C_C$）。选择 L 的最优坐标使得卡车行驶总距离最短，无须考虑行驶时间，输出卡车的最短行驶距离。